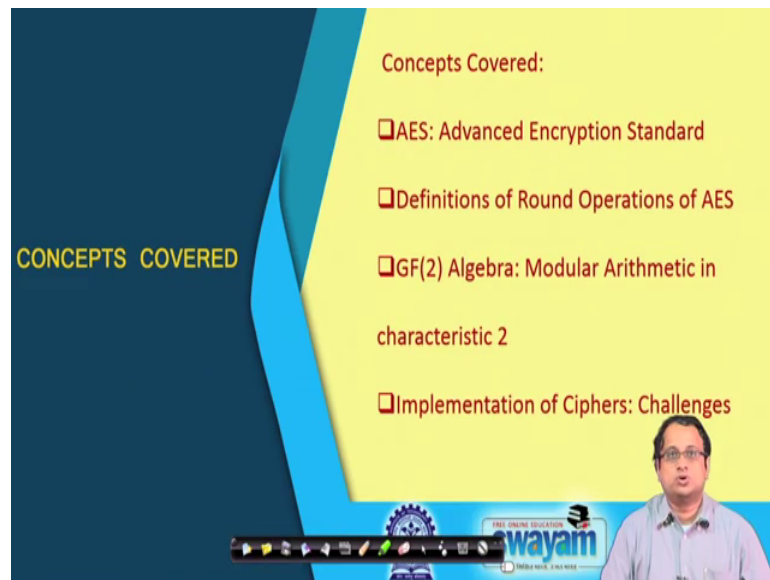


Hardware Security
Prof. Debdeep Mukhopadhyay
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 09
Advance Encryption Standard (AES) and Side Channel Analysis

So welcome back, so today we shall be trying to understand about or looking into more details about the Advanced Encryption Standard, which we started to discussed in the last class. And, we shall also talks upon the definition of side channel analysis which is so important for hardware cryptography.

(Refer Slide Time: 00:31)



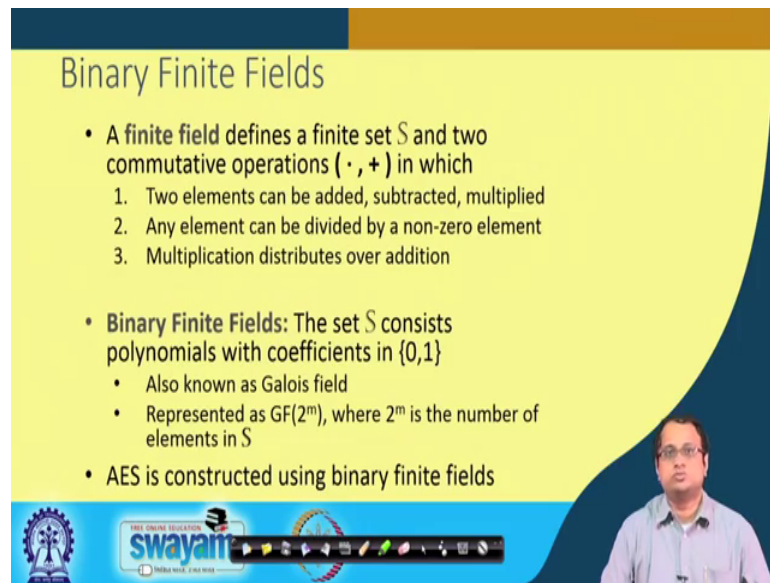
The slide is titled "CONCEPTS COVERED" and lists the following topics:

- ❑ AES: Advanced Encryption Standard
- ❑ Definitions of Round Operations of AES
- ❑ GF(2) Algebra: Modular Arithmetic in characteristic 2
- ❑ Implementation of Ciphers: Challenges

The slide also features a video feed of the professor in the bottom right corner and a navigation bar at the bottom.

So, the concepts that we shall be covering in today's class is that we shall be trying to look into the AES blocks. So, we shall be trying to go through the definitions of the various round operations in AES and in this context are very important algebra which we need to look into is what is called as GF 2 algebra or Galois field 2 algebra, where we will trying to look into how modular arithmetic is done in characteristic 2 fields; so and finally will be trying to conclude with some challenges on Cipher implementations.

(Refer Slide Time: 01:01)



The slide is titled "Binary Finite Fields" and contains the following text:

- A finite field defines a finite set S and two commutative operations $(\cdot, +)$ in which
 1. Two elements can be added, subtracted, multiplied
 2. Any element can be divided by a non-zero element
 3. Multiplication distributes over addition
- Binary Finite Fields: The set S consists polynomials with coefficients in $\{0,1\}$
 - Also known as Galois field
 - Represented as $GF(2^m)$, where 2^m is the number of elements in S
- AES is constructed using binary finite fields

The slide also features a presenter in the bottom right corner and a navigation bar at the bottom with the Swamyam logo.

So, just to begin with binary fields or binary finite fields essentially are essentially are finite fields which has got which is denoted, for example by the set S and there are two commutative operators which are define. So, it is typically denoted by set dot and a plus. So, a dot is of an it kind of realize as a multiplication and a plus is with additions, but essentially they can the definitions can be you know like varied and the definitions are just indicative of this of 2 operations.

So, so therefore right you can basically take 2 elements in this field and you can you know like add subtract and multiply and you can also divide by any non zero element ok. So therefore, idea is that every element here which is non zero should have a multiplicative inverse and therefore you should be able to multiply with the multiplicative inverse. And there is another a very important rule which is called as distribution, so the multiplication will should distribute over addition ok. So therefore, right in binary finite fields which is the very I mean the smallest form of the binary finite field will typically have only 2 elements which is 0 and 1 and that is also called as $GF(2)$ or Galois field 2 ok. So, this is usually called as Galois field 2 or $GF(2)$.

Now, there is an extension of this field which is called as $GF(2^m)$ power of m , which is essentially nothing but an m bit extension of $GF(2)$. So therefore, here the 2 to the power of m is often the number of elements in S and I the you know like why it is very important to study you know like $GF(2)$ based arithmetic or $GF(2)$ based algebra, is

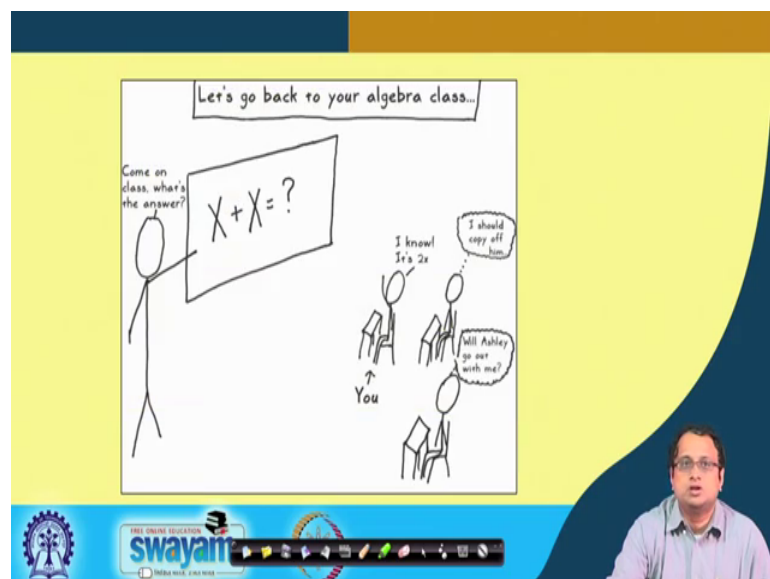
because of the fact that various cryptographic algorithms; like for example, AES is reliant or constructed using binary finite fields and partially it is done because of its efficiency like you know that. If you for example, have a GF 2 arithmetic then you basically can have only a single bit to indicate that data you can either be 0 or it can be either be 1.

GF 2 power of m element can be realized likewise by an m bit register and therefore you essentially have a very compact representation of an element in this finite field. So, and also right as we will study is that when you add as we have also started in our previous discussions is that when you have got GF 2 arithmetic then many of the underlying arithmetic becomes efficient implementation.

For example, if you remember you know like when we are doing addition then GF 2 base additions can be done only by exclusive or's ok. Likewise a squaring was very easy to implement you just needed to interpose 0 in between and you essentially had this basic squaring done right. Of course, you have to do the final modular reduction and also the final modular reduction when you are doing a modular with an irreducible polynomial can also very efficiently be replaced would be realized by only using exclusive or's ok.

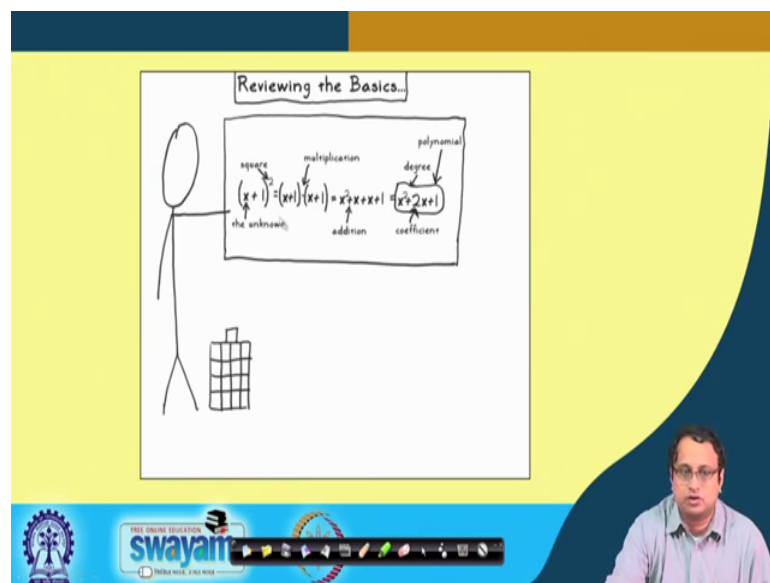
So therefore, write binary finite field bit circuits are amenable to efficient implementations and that is also one of the reasons why we have got very efficient architectures for the AES algorithm.

(Refer Slide Time: 04:03)



So therefore, write I mean if you really want to understand again I will be deserting to the stick diagram which I cited in my last class ok. So, if I want to for example, you know like look into this then there are certain weird things which happens in GF 2 based arithmetic which we have tried we have which we have to get used to works ok. For example, suppose if I ask you right how what is X plus X ok, so the usual answer could be 2 X right. But in Galois field arithmetic when you are doing X plus X since the plus is nothing but exclusive or you essentially have 0 ok, so that is the very weird result which you get.

(Refer Slide Time: 04:41)



So, likewise right I mean there are certain things which will which will change when you are essentially doing it algebra in GF 2 arithmetic. So, it is it is important to get used to this difference in algebra, for example if I ask you to calculate x plus 1 squared the usual answer is x squared plus 2 x plus 1, but if when you are doing in GF 2 then this 2 x is 0 ok, so because 2 is 0 because you are doing modulo 2 arithmetic. So therefore, the result will be x squared plus 1 ok. So therefore, x plus 1 whole square is nothing, but x squared plus 1 likewise x plus 1 whole to the power of 4 will be not only x to the power of 4 plus 1.

(Refer Slide Time: 05:19)

We'll change things slightly. In the old way, coefficients could get as big as we wanted. In the new way, they can only be 0 or 1:

Old Way
 $123x^2 + 45x + 678x + 9x + 10$
 $= 168x^2 + 687x + 10$
Big coefficients

New Way
 $x^2 @ x @ x @ x @ x @ 1$
 $= x^2 @ 1$ The 'new' add'
Small coefficients
 $x^2 @ x @ x @ (x^2 @ x^2) @ x^2$
 $= 0 @ x^2$
 $= x^2$

*Nifty Fact: In the new way, addition is the same as subtraction (e.g. $x @ x = -x = 0$)

So therefore, right I mean the things will change slightly, therefore in the in the old ways when we are for example having you know when we are doing our computations you know like then essentially what happens is that suppose you are doing you know like say multiplication of 2 polynomials or if you are adding 2 polynomials right, then typically the things keep on increasing ok. But when you are essentially operating on finite fields as the name suggests it is finite, you have to do something is called as a modular operation right. So, that the essentially the field essentially has got finite number of elements.

So therefore, right usually as we have also discussed probably previously is that usually these elements of the field or the finite field are represented as polynomials ok. So, these polynomials are the x and the coefficients are part of the polynomial are taken in the base field. That means, like if I take a field of $GF 2$ to the power of m then an element in $GF 2$ to the power of m can be represented by an m bit register or equivalently in terms of polynomial it can be represented by a polynomial of degree m minus 1 ok. So, therefore, it can be increase from the degree can be from a constant term 2^m minus 1 and the coefficients of the polynomial are elements of $GF 2$, that means they can be either 0 or they can be either 1.

So, when you want to operate on 2 such polynomials and suppose you want to multiply 2 such polynomials there is always a chance that the degree you will exceed m minus 1

right. So, if you want to bring it back to the field therefore you need to do an modulo operation and the modulo operation is done usually by applying or taking an irreducible polynomial which has got degree of m . And therefore once you do a reduction; that means, you divide it and take the remainder that means you take the resultant polynomial divide by this irreducible polynomial which has got degree of x of m then that you would imply the remainder has always degree which is less than m ok. That means, the degree as the degree of the remainder is always maximum m minus 1 and therefore the remainder belongs to the field.

So, if you understand this right essentially you are basically understood the crux of GF 2 arithmetic or finite field in general. So, what essentially for example, you know like when you are so there are small differences right as I said which we have to get used to. For example, right if in the old way right when you are doing a computation, suppose you are doing a computation, like you know like in the old way the coefficients could be as big as possible. Suppose when you are doing some arithmetic I have got a coefficient when you say $123x^2 + 45x^2 + 678x + 9x + 10$, so therefore, you can see that in all the way right or usual algebra we essentially often see huge or large number of coefficients.

But in the new way that means, when you are doing these computations here you have to reduce these coefficients ok. So, you have to reduce this coefficient GF 2 that means, if you for example do it or reduce it here or apply it here then 168 will be 0 because, that is an even number 687 will be odd so it is 1 likewise 10 will be 0 ok. So, if therefore, you will be left with only x .

So therefore, the in the in this you know in this algebra you will always have terms like $x^2 + 1$ or $x^2 \oplus 1$ also alternatively as it is written or we will have x^2 and so on ok. So for example, here if I want to do say x^2 , so here as it is shown here right I mean you can see that I have got $123x^2$. So, the $123x^2$ is being written as x^2 that is 1 into x^2 , likewise $45x^2$ is nothing but 1 into x^2 $678x$ is you know like $678x$ as you can understand this is an even number, so therefore this should be it should go away ok.

A $9x + 10$ will be again $x + 1$ because this is essentially your odd number ok. So, so likewise you can do this simplification and finally you will be left with small

coefficients. So, finally here you have got only say you know like small coefficients then the coefficients belong to either 0 or 1. So, one important observation is also that in the new way or the new definition addition is same as subtractions. So, therefore if you are doing say $x \text{ xor } x$ that is same as doing $x \text{ minus } x$ and that is 0. So therefore, addition and subtraction are same in both the cases they are equivalent.

(Refer Slide Time: 09:37)

We use our friend, 'clock math', to do this.
Just add things up and do long division.
Keep a close watch on the remainder:

4 o'clock + 10 hours = 2 o'clock

\Rightarrow 4 + 10 hours = 2

\Rightarrow 4 + 10 = 14

\Rightarrow
$$\begin{array}{r} 12 \overline{)14} \\ \underline{12} \\ 2 \\ \hline \end{array}$$

*This is also known as 'modular addition.' Math geeks call this a 'group.' AES uses a special group called a 'finite field.'

So therefore, write here these arithmetic is often understood by a clock arithmetic. So, it is a it is basically modular arithmetic for example, there is 4 o clock if I add 10 hours then we will say it is 2 o' clock because, we are doing implicitly a modular 12 operation ok. So, likewise right when you are this is essentially called as modular addition and you know like in typically we also call that as developing a special group which is called as a finite field.

(Refer Slide Time: 10:05)

Remember how multiplication could make things grow fast?

$$(x^7 + x^5 + x^3 + x)(x^6 + x^4 + x^2 + 1)$$

$$= x^{13} + x^{11} + x^9 + x^7 + x^5 + x^3 + x$$

Big and yucky!

So, therefore, right it is so we can probably take this example to understand the advantage of such kind of modular operations, suppose I take a polynomial x to the power of 7 plus x to the power of 5 plus x cube plus x note that this polynomial is in GF_2 to the power of 8. Because, as I said that when you consider GF_2 to the power of 8 then that would imply a polynomial which has got a degree of maximum 7 and therefore this belongs to GF_2 to the power of 8.

Let me take another polynomial in GF_2 to the power of 8 say x to the power of 6 plus x to the power of 4 plus x squared plus 1 and then multiply them if I multiply them of course, I will get terms which has got degrees say 7 plus 6 that is x power of 13. Now you can see that this final result right is quite big, so for example I have got x to the power 13 plus 2 x to the power of 11 plus 3 x to the power of 9 and 17 plus x . So therefore, when you are doing modular arithmetic we would like to reduce it and bring it back to the field. And therefore you need a polynomial which is called as an irreducible polynomial and the other thing which you also need to ensure is that these coefficients are in GF_2 .

(Refer Slide Time: 11:13)

We can do 'clock' math with polynomials. Instead of dividing by 12, my creators told me to use $m(x) = x^8 + x^4 + x^3 + x + 1$. Let's say we wanted to multiply $x \cdot b(x)$ where $b(x)$ has coefficients b_0, \dots, b_7 :

$$x \cdot b(x) = x \cdot (b_7 x^7 + b_6 x^6 + b_5 x^5 + b_4 x^4 + b_3 x^3 + b_2 x^2 + b_1 x + b_0)$$

$$= b_7 x^8 + b_6 x^7 + b_5 x^6 + b_4 x^5 + b_3 x^4 + b_2 x^3 + b_1 x^2 + b_0 x$$

⚡ Eeek! x^8 is too big. We must make it smaller.

* Remember that each b_i (e.g. b_7) is either 0 or 1.

So therefore, what we do is that we try to you know like take a polynomial for example, a usual polynomial here would be say x to the power of 8 plus x to the power of 4 plus x to the power of 3 plus x plus 1 and then we want to ensure that the result is brought back to the field. So, again to understand how we do this modular operation let us take a very simple operation ok, let me take a $b \cdot x$ where $b \cdot x$ is nothing but $b \cdot 7 x$ to the power of 7 plus $b \cdot 6 x$ to the power of 6 plus $b \cdot 5 x$ to the power of 5 plus $b \cdot 4 x$ to the power of 4 plus $b \cdot 3 x$ to the power of 3 plus $b \cdot 2 x$ square plus $b \cdot 1 x$ plus $b \cdot 0$ and then I multiplied with x .

Note that if $b \cdot 7$ is 1 then I get $b \cdot 7 x$ to the power of 8, now the moment I get $b \cdot 7 x$ to the power of 8 I know that this element does not belong to the field because, it takes the degree exceeds 7. So therefore, we have to basically make it smaller and we have to bring it back to the field and therefore the usual way of doing that is by taking this polynomial and being dividing by this polynomial which is $m \cdot x$ to the power of 8 plus x to the power of 4 plus x to the power of 3 plus x plus 1 now this is an irreducible polynomial ok.

(Refer Slide Time: 12:21)

We divide it by $m(x) = x^3 \oplus x^2 \oplus x \oplus 1$ and take the remainder:

$$\begin{array}{r} x^7 \oplus x^3 \oplus x \oplus 1 \\ \oplus x^3 \oplus x^2 \oplus x \oplus 1 \\ \hline x^4 \oplus 1 \end{array}$$

Remainder $\rightarrow b_2x^2 \oplus b_1x \oplus b_0$

Note how the b's are shifted left by 1 spot. This is just b_i multiplied by a small polynomial.

So, what I do is that this is something like I do a modular arithmetic like what we do in modular arithmetic, here also when we work with polynomials we do exactly the same. So, I take this polynomial divide the by this polynomial and I get the remainder the final remainder is being shown here and this remainder essentially belongs to the field ok.

(Refer Slide Time: 12:49)

Why bother with all of this math? Encryption deals with bits and bytes, right? Well, there's one last connection: a 7th degree polynomial can be represented in exactly 1 byte since the new way uses only 0 or 1 for coefficients:

$$\begin{array}{r} x^7 \oplus x^3 \oplus x \oplus 1 \\ = 0x^7 \oplus 0x^6 \oplus 0x^5 \oplus 1x^4 \oplus 1x^3 \oplus 0x^2 \oplus 1x \oplus 1 \\ = \begin{array}{ccccccc} 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{array} \\ \leftarrow \text{hexadecimal } 10110101_2 = b_7 \leftarrow \text{hexadecimal} \\ = 1b_{16} \text{ A single byte!} \end{array}$$

*Although we'll work with bytes from now on, the math makes sure everything works out.

So therefore, the object the final the why it belongs to the field because, you can see that the maximum degree here is 7 ok, therefore this belongs to the field. So, therefore, right I mean so therefore you know like what you can do is that rather you know like, I mean

rather working with polynomials when you are trying to think about implementations a better way is to realize that is in the form of register.

So, what you can probably do is that suppose I have got a polynomial like x to the power of 4 plus x to the power of 3 plus x plus 1 I can represent them by a polynomial by a register which has got 8 elements ok. For example, here the elements would be 0 here because this degree is 0 here it does it is not there in this particular element, likewise x to the power of 6 will have a coefficient of 0 x to the power of 5 will have a coefficient of 0 x to the power of 4 will have a coefficient of 1.

Likewise we will have x^3 plus 0 x^2 plus 1 x plus 1 ok. So therefore, this number is nothing but 0 0 0 1 and that therefore in hexadecimal I can denote it as 1 and here I have got 1 0 1 1 in hexadecimal this is b ok. So therefore, right I can actually represent this by a single byte 1 byte which is denoted by this hexadecimal notation which is 1 b, so therefore 1 b also stands for this polynomial. So therefore, what we will probably try to write now is that we will try to work with say these numbers 1 b and when we are operating with 1 b whether we are multiplying adding write will basically implicitly we will be doing finite field operations. But this gives a very nice compact representation of GF 2 power of 8 element.

(Refer Slide Time: 14:21)

Since we know how to multiply, we can find the 'inverse' polynomial byte for each byte. This is the byte that will undo/invert the polynomial back to 1. There are only 255 of them, so we can use brute force to find them:

$$(x^4 + x^3 + x + 1) \cdot ? = 1$$

1b (hex) = 1
found using a brute force for-loop

* There are only 255 instead of 256 because 0 has no inverse.

So therefore, right when we are trying to say you know like when we are trying to construct a field a very important property of being a field is that every nonzero element

should have a multiplicative inverse, that means suppose if I give you this one b which is essentially standing for this polynomial there should be an element with which if I multiply I get back the multiplicative unity which is 1 ok, which is nothing but the polynomial with one single constant that is one. So therefore, right I should be able to find out this corresponding element which if I multiply I should get back a 1 ok.

There are different ways as we have started in the last class of computing multiplicative inverses you can apply Euclidean algorithm or any other algorithm for that matter, but you should also understand that when you are you know like realizing it for a small field like for a AES which is realized on GF 2 power of 8 arithmetic, you do not need to apply the very general algorithm for computing the inverse ok.

So, one very nice way would be let me store all the inverses, let me store all the multiplicative inverses for non 0 elements and that would mean that I need only 255 storages right. Because, I need storages for 255 elements and I can get the corresponding inverse in order one time right, I can just get directly the inverse and I can do that with a small amount of storage.

(Refer Slide Time: 15:43)

Now we can understand the mysterious s-box. It takes a byte 'a' and applies two functions. The first is 'g' which just finds the byte inverse. The second is 'f' which intentionally makes the math uglier to foil attackers.

$g(a) = a^{-1}$

$f(a) = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_7 \\ a_6 \\ a_5 \\ a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$

sbbox[a] = f(g(a))
 sbbox[5B] = f(g(5B))
 sbbox[5B] = f(g) = 6A
 5B^-1 = 01

The slide also features a logo for 'swayam' (The Online Education) and a small video inset of a man speaking in the bottom right corner.

So, therefore, right I mean let us try to understand how the substitution box or the s box of AES looks like ok. So, if you have understood how do or rather the idea of multiplicative inverse then the representation of the s box becomes very easy. So, what you do here is by 2 steps so therefore, suppose that in the s box you essentially have an

input a and if I apply g of a then I get a inverse which is the multiplicative inverse and likewise the next step which you do after computing $g^{-1}(a)$ is to apply an affine transformation.

So, the affine transformation is done by taking the result of a inverse which is denoted as a^{-1} , note that this registered stands for an element in $GF(2^8)$ and then I apply a matrix M pre multiply with the matrix which is fixed and then I add another matrix v . So therefore, I get you know like a corresponding result and therefore the final result is essentially the corresponding output of this affine transformation. So, for example therefore, if I want to calculate says the x box of 58 , so note that 58 stands for the hexadecimal notation of 5 followed by 8 . So, it will be basically 0101 that stands for 5 and likewise your 8 will be 1000 so 1 triple 0 .

So therefore, if I take this then first I will calculate the multiplicative inverse of this which is essentially here 18 and you can calculate that if I multiply 58 with 18 and then apply the modular polynomial that modular polynomial which is essentially $x^8 + x^4 + x^3 + x + 1$ then I should get back one I should get back one here and finally right I apply that the affine transformation which is shown by this matrix plus this vector and then I get the result which is $6a$, so that means, 58 we will get mapped into $6a$.

So, this mapping is predetermined in case of AES, that means this s box either you can you know like pre calculate this and store in the form of a table and the size of the table as you can easily understand will be our dimension 256 because, there are 2^8 possible inputs that you can give to this s box it operates on 1 byte of information.

(Refer Slide Time: 17:55)

Mix Columns is the hardest. I treat each column as a polynomial. I then use our new multiply method to multiply it by a specially crafted polynomial and then take the remainder after dividing by x^4+1 . This all simplifies to a matrix multiply:

$$b(x) = c(x) \cdot a(x) \pmod{x^4+1}$$

special polynomial the column

$$x^4+1 \mid \begin{matrix} 03a_3x^3+03a_2x^2+03a_1x+03a_0 \\ +01a_3x^3+01a_2x^2+01a_1x+02a_0 \\ +01a_3x^3+01a_2x^2+01a_1x+02a_0 \\ +02a_3x^3+02a_2x^2+02a_1x+02a_0 \end{matrix}$$

$$\begin{matrix} 03a_3x^3+03a_2x^2 \\ 3a_3x^3+3a_2x^2+3a_1x+a_0 \\ +2a_3x^3+2a_2x^2+2a_1x+2a_0 \\ 3a_3x^3+a_2x^2+3a_1x+a_0 \\ 3a_3x^3+3a_2x^2+a_1x+a_0 \\ +3a_3x^3+3a_2x^2+a_1x+a_0 \\ +2a_3x^3+2a_2x^2+2a_1x+2a_0 \\ +3a_3x^3+3a_2x^2+a_1x+a_0 \\ +3a_3x^3+3a_2x^2+a_1x+a_0 \\ +2a_3x^3+2a_2x^2+2a_1x+2a_0 \\ +3a_3x^3+3a_2x^2+a_1x+a_0 \end{matrix}$$

$$\begin{pmatrix} 2 & 1 & 3 \\ 3 & 2 & 1 \\ 1 & 3 & 2 \\ 1 & 1 & 3 & 2 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

Mix column is probably the more you know like complicated step where you basically take you know like each of these columns. So, in AES as I say that in AES 128 a state is represented as in this form so it is represented as a matrix; so, this matrix has got 16 bytes ok. So, you can either visualize them as sixteen bytes or you can visualize them to be made of 4 columns and each column has got dimension of 32 bits or 4 bytes. So, in mix columns what you do is you take one of these columns and you transform them and get another output column and how do you do this transformation is by taking this column and writing them as another matrix a x or another polynomial say a x, but a x now has got elements such that you know like I can write ax as a 3 x to the power of 3 plus a 2 x square plus a 1 x plus a 0, note that all these coefficients a 0 a one a 2 and a 3 now belongs to GF 2 power of 8.

And now what I do is that I multiply it with a fixed polynomial which is a special polynomial and then I take modulo x to the power of 4 plus 1 ok. Note that x to the power of 4 plus 1 is not an irreducible polynomial in this field and therefore all elements will not have multiplicative inverse. But at the same time if I want to apply you know to get the decryption I need to ensure that this special polynomial has a multiplicative inverse ok. So therefore, it indeed happens right that for this particular polynomial or reducible polynomial this polynomial has a multiplicative inverse and therefore right we can apply a decryption process.

So, in the encryption step what we do is that we take this polynomial, we multiply it with this special polynomial and then apply modulo x to the power of 4 plus 1 to get another result which essentially. Also will therefore, have you know like can be represent it as shown here x^3 with you know like degrees like x^3 x^2 x and constant term where the coefficients are as shown here $2a^3$ plus a^2 plus a plus $3a^0$ for x^2 it will be $3a^3$ plus $2a^2$ plus a plus 0 likewise for x it will be $3a^3$ plus $3a^2$ plus $2a$ plus 0 . Note that these operations like $2a^3$ $3a^0$ they are done in GF 2 power of 8 they are not integer multiplications ok.

So therefore, when you are doing 2 into a 3, that means you are doing you are multiplying 2 which stands for the polynomial x with a 3 which is the element in G of 2 power of 8. That means, you can represent is a 3 has to be made of you know like a register which has got 8 you know register of dimension 8 and each of them are 0 1 values or you can imagine on them to be as a polynomial we just got a degree of maximum 7 and therefore when you multiply it with x there is always a term overflow. The moment there is an overflow again you apply the irreducible polynomial as I said in the previous slide and you bring the result back to GF 2 power of 8 and finally you essentially try to compose them and you essentially get this coefficient.

Alternatively there is a very similar simple representation of this transformation which is done through this mixed column matrix. So, here you take this column which is denoted as $a^3 a^2 a^1 a^0$ and imagine that you are basically pre multiplying with this matrix. So, this matrix essentially has got elements 2 1 1 3 and then there is a cyclic shift of these elements, you can see that I get 3 2 1 1 1 3 2 1 1 1 3 2 again note that all these elements belong to GF 2 power of 8. So therefore, the multiplication is done in Galois field 2 to the power of 8 and finally you get the result which is shown as $b^3 b^2 b^1 b^0$ which is nothing, but the transformed column ok. So, you apply this column by column and you essentially transform the entire state matrix and you get the result after the Mix column operation.

(Refer Slide Time: 21:49)

Add Round Key

a	e	i	m
b	f	j	n
c	g	k	o
d	h	l	p

 \oplus

k_0	k_4	k_8	k_{12}
k_1	k_5	k_9	k_{13}
k_2	k_6	k_{10}	k_{14}
k_3	k_7	k_{11}	k_{15}

 \rightarrow

$a+k_0$	$e+k_4$	$i+k_8$	$m+k_{12}$
$b+k_1$	$f+k_5$	$j+k_9$	$n+k_{13}$
$c+k_2$	$g+k_6$	$k+k_{10}$	$o+k_{14}$
$d+k_3$	$h+k_7$	$l+k_{11}$	$p+k_{15}$

So therefore, right let us take again a you know like let us summarize these steps what we have seen, the first step is the add round key. So, then add round key you take a state of the AES 128 we have got a key, again this key is nothing but a 16 byte data which is again represented by this tabular representation where every element is a byte and then you do an exclusive or this is your key mixing step and you get the final result here this plus stands for exclusive or when you are doing a bitwise exclusive or.

(Refer Slide Time: 22:19)

Shift Rows

a	e	i	m
b	f	j	n
c	g	k	o
d	h	l	p

 \rightarrow

a	e	i	m
f	j	n	b
k	o	c	g
p	d	h	l

- **ShiftRows**
 - Leave the First row untouched
 - Left Rotate (2nd Row by 8 bits)
 - Left Rotate (3rd Row by 16 bits)
 - Left Rotate (4th Row by 24 bits)
- **Implementation in Hardware**
 - No resources required, only mapping with wires

Then there was an important step which is called as shift rows, in shift rows you take this particular state matrix and you will see that in the first row I do not do any shift ok. So therefore, if the first row is a e i m, I essentially have got a e i m in the output of this transformation as well. But if you see the next row then for example, b f j n then you will see that I do a cyclic left rotation ok. So therefore, b comes here f comes here j comes here and n comes here. So, I do is so cyclic left rotation by 8 bits that means 1 by 1 byte, for the third row I do a cyclic left rotation but now I do it by 2 bytes. So therefore, you know like k will come here and so on, so I will get k o c g in this case.

The final row essentially is done again a left rotation but now it is by 3 bytes ok, you can orderly think of this as a right rotation by one byte as well ok. So therefore, d h l p will therefore, come here as d will come here h will come here l will come here and p will come here. So finally, you can also realize that if you want to realize that by a hardware then you do not need to waste any resource because, you can do this entire mapping by just wirings you do not need any explicit step for doing this computation you can just do a wiring and therefore without any resource you can apply you can achieve this transformation ok.

(Refer Slide Time: 23:43)

Inverse Shift Rows

a	e	i	m
f	j	n	b
k	o	c	g
p	d	h	l

→

a	e	i	m
b	f	j	n
c	g	k	o
d	h	l	p

- **ShiftRows**
 - Leave the First row untouched
 - Right Rotate (2nd Row by 8 bits)
 - Right Rotate (3rd Row by 16 bits)
 - Right Rotate (4th Row by 24 bits)

The wiring diagram on the right shows the mapping between the original and shifted rows. The original rows are a, f, k, p, e, j, o, d, i, n, c, h, m, b, g, l. The shifted rows are a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p.

Inverse shift row can be again done the similar way this is just the opposite operation and which you can again do by a wiring operation.

(Refer Slide Time: 23:51)

Mix Columns

$$\begin{aligned}
 E &= 2e + 3f + g + h \\
 F &= e + 2f + 3g + h \\
 G &= e + f + 2g + 3h \\
 H &= 3e + f + g + 2h
 \end{aligned}$$

- The 4x4 matrix is multiplied with the matrix

$$\begin{bmatrix}
 2 & 3 & 1 & 1 \\
 1 & 2 & 3 & 1 \\
 1 & 1 & 2 & 3 \\
 3 & 1 & 1 & 2
 \end{bmatrix}$$

swayam

Finally, you have got the mixed columns as I say that is for the Mix columns. So, you basically have to realize this column transformation where you take this column apply this pre this matrix. So, this matrix is what I just now described, where you have got elements in GF 2 to the power of 8 and you have to multiply or pre multiply this matrix with this column to get the final result shown here as capital E capital F capital G and capital H ok.

(Refer Slide Time: 24:15)

Inverse Mix Column

$$\begin{aligned}
 E &= Ee + Bf + Dg + 9h \\
 F &= 9e + Ef + Bg + Dh \\
 G &= De + 9f + Eg + Dh \\
 H &= Be + Df + 9g + Eh
 \end{aligned}$$

- The 4x4 matrix is multiplied with the matrix

$$\begin{bmatrix}
 E & B & D & 9 \\
 9 & E & B & D \\
 D & 9 & E & B \\
 B & D & 9 & E
 \end{bmatrix}$$
- The hardware implementation can be done in a similar way as mix columns

swayam

So, likewise I say that the matrix is chosen in a way, so that it has got an inverse and therefore this is the corresponding inverse which you can see is slightly more complicated than the forward transformation and then one once you have got this right you essentially can again pre multiply with the result and you should get back the original data.

So, here as you can see that I have taken this input and again got back this, so this give me back the original data ok. The hardware implementation can be done in a similar way as the mix columns and we will be studying this in the next class of more specific details about these implementations.

(Refer Slide Time: 24:49)

Byte Substitution

- Makes a non-linear substitution for every byte in the 4x4 matrix

Sbox

a	e	i	m
b	f	j	n
c	g	k	o
d	h	l	p

A	E	I	M
B	F	J	N
C	G	K	O
D	H	L	P

Affine Transformation

$$Sbox(x) = \begin{cases} \text{Affine}(x^{-1}) & \text{if } a \neq 0 \\ \text{Affine}(0) & \text{if } a = 0 \end{cases}$$

a_7	1	1	1	1	0	0	0	a_0	0
a_6	0	1	1	1	1	0	0	a_1	1
a_5	0	0	1	1	1	1	0	a_2	1
a_4	0	0	0	1	1	1	1	a_3	0
a_3	1	0	0	0	1	1	1	a_0	0
a_2	1	1	0	0	0	1	1	a_1	0
a_1	1	1	1	0	0	0	1	a_0	1
a_0	1	1	1	1	0	0	0	a_1	1

Finally as I say the byte substitution or the s box which provides confusion is a non-linear transformation, so here you do and I find you do an x inverse computation in GF 2 to the power of 8 followed by an affine transformation for 0 as an input. Then you assume that 0 is the corresponding inverse of it as the inverse is not defined, we define we make a special case for that and say that the inverse of 0 is 0 and then I do a final find transformation on that. And, finally this is the affine matrix as shown here these matrices are vectors are predefined and pre design and therefore I can define the transformation for AES s box or by it substitution.

(Refer Slide Time: 25:27)

S-box Encryption Table

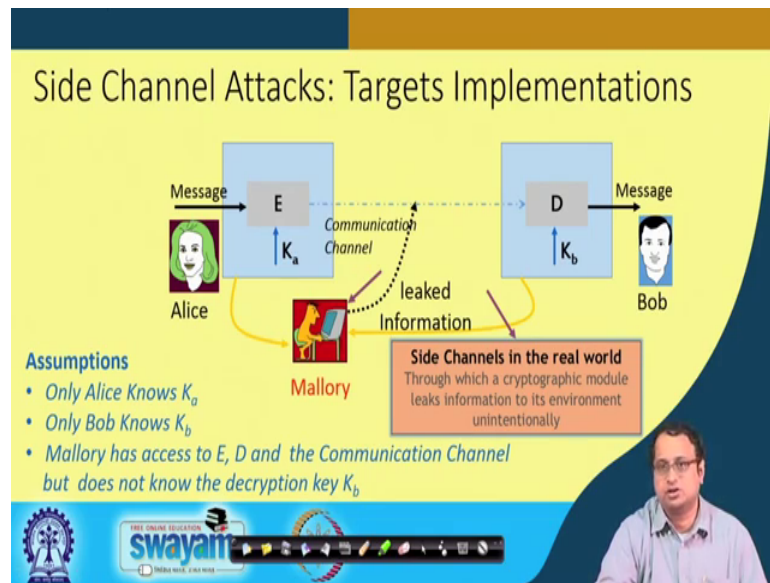
- Use a table to do the byte substitution
- Eg. $S_{\text{box}}[42] = 2c$

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	9f	7b	f2	6b	6f	c5	30	01	67	2b	f6	d7	ab	76
1	ca	82	c9	7d	fa	59	47	e0	ad	d4	a2	a7	3c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f3	71	d8	31	15
3	04	c7	23	c3	13	96	05	9a	09	12	80	a3	ab	27	32	75
4	03	f1	c4	1a	1b	6a	a0	52	3b	d6	b3	29	a1	2f	04	
5	53	d1	00	ed	20	2c	b1	5b	4a	cb	39	4a	4c	58	c7	
6	d0	ed	ak	eb	43	4d	33	05	45	f9	02	72	50	3c	5f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b4	da	21	10	ff	f3	d2
8	cd	0c	13	ac	5f	97	44	17	e4	a7	7e	3d	04	5d	19	73
9	00	81	42	dc	22	2a	30	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	ea	79
b	e7	e8	17	6d	8d	d5	4e	a9	6c	56	f4	ea	e5	7a	aa	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	35	66	48	03	f6	0e	61	35	57	b9	0c	1d	9e	
e	e1	f9	08	11	69	d9	8e	34	9b	1e	07	a9	ce	55	28	d7
f	8c	a1	89	08	b7	ea	42	e8	41	39	2d	0f	b0	54	bb	16

So finally, what I do is that for example, if I want I can store this in the form of a table. So, what I can for example suppose my input is 4 2. So, what I do is in that table I have got you know like a tabular representation where the rows are indexed by numbers from 0 to f and the columns are also indexed from 0 to f.

So therefore, if 4 2 is my input then I see the corresponding row here and I could see the corresponding column here and therefore 2 c is my result ok. So therefore, I can also realize this by a nice in a nice tabular representation, although this may not be the best way of the implementing it. But at least we understand how we can you know like possibly realize such kind of mapping. So, we will be studying in the next class about more details about how to realize this s box operation.

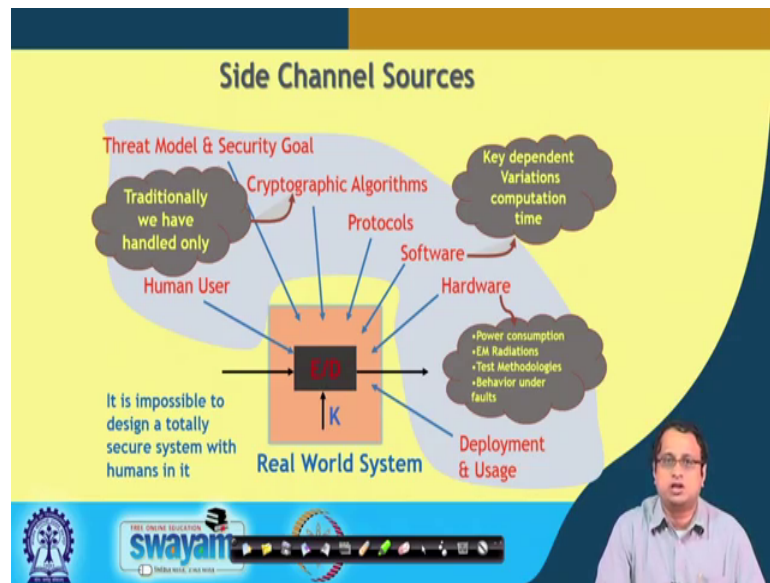
(Refer Slide Time: 26:13)



So, talking about like finally our goal is to develop a hardware design, but one of the very important you know like. So, it basically in the in the classical sense right what we have discussed in the previous discussion or lecture also that you have got Alice and Bob and you have got an attacker which is either eve or Mallory or some eavesdropper who is trying to observe this communication channel. So, therefore, what we essentially assumed in the classical sense was that only Alice knows k_a only Bob knows K_b and Mallory has got access to the encryption algorithm decryption algorithm and the communication channel, but does not know the decryption key ok . So, this is the classical scenario.

But in real life when you have either a hardware design or if you have an embedded design or you have any implementation for that matter, then the attacker can try to you know like look into several other information sources and these are what are called as side channels. So, in the side channels in the real world essentially scanned for those unintentional or even you know like unintentional information leakages which can happen in the real world and can compromise your secrecy ok , can compromise the information about the secret key.

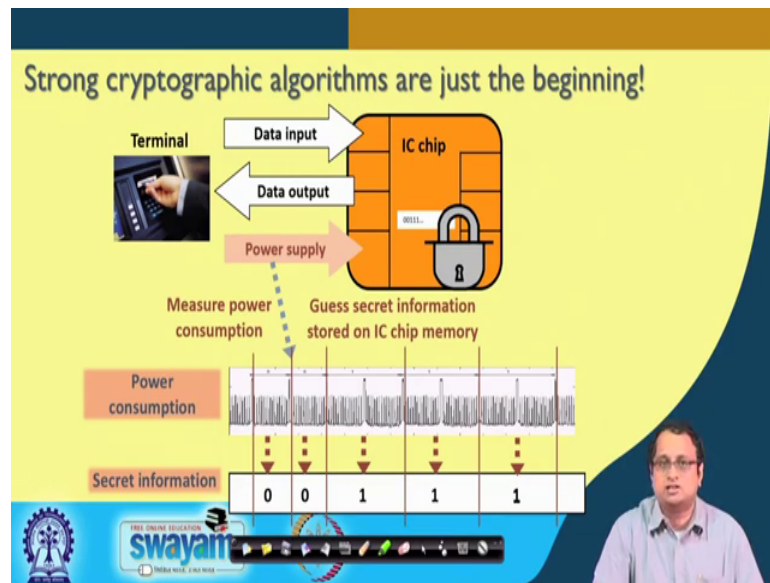
(Refer Slide Time: 27:19)



So therefore, there can be several side channel sources ok, so cryptographic algorithms there are you know like the threat models and the security goal is also important ok, the protocol is important the software is important whether you are doing a hardware design is important and finally your deployment and usage is also very crucial and of course you have the human users and it is almost impossible to design it orderly secure system with humans in it ok.

But rather right what we will be trying to look into is non those aspects, but more you know like as we have traditionally been handling only cryptographic algorithms, in this course we will shall be trying to look into more of you know that the hardware and the software leakages. For example, in software you can have key dependent variations computation times and so on which you will try to see how we can exploit and also suitably safeguard against and in hardware of course you have got several side channel sources like power consumptions EM radiations test methodologies behavior under faults which all can lead to efficient mechanisms of attacks. And, therefore when you want to really make secure hardware you need to take care of these threats scenarios and properly address them.

(Refer Slide Time: 28:27)



So therefore, right the fundamental claim here is that strong cryptographic algorithms are important, but they are just the beginning. So, it may you know in this scenario for example, take a smart card and take it to a card accepting device have a nice encryption algorithm. So, when you are just restricted to only input and output exchanges then this may be safe ok. But in the real world right there may be for example, power supply is it and that power supply can previously leak the key and therefore right for example like if you have an if you have studied.

For example, RSA algorithm right the RSA algorithm does a square and multiply operation and therefore what the happen is that from the power consumptions here depending upon the glitches or stressing the glitches of the power rail you can probably infer that the secret key was 0 ok, we are and if the glitch is more or this is more then you see that probably it is doing an additional step and the additional step is probably a multiplication and therefore it will give you the secret key you know order n amount of time you need an amount of time.

So therefore, you are not really challenging the mathematics behind RSA, but you are just having an very easy way of getting the key, but your target is mainly the implementation and therefore it also ensure that our implementation should not be efficient as well only, but at the same time also should try to restrict again these kind of attack vectors.

(Refer Slide Time: 29:43)

What are Side Channels?

- These are covert channels which leak information which the designers of cryptographic algorithms did not consider.
- Information is leaked because of the implementation:
 - optimization leads to information leakage
 - example: **an if-else statement in a programming language**

The slide features a yellow background with a dark blue curved shape on the right. At the bottom, there is a blue banner with the Swamy logo and a video feed of a man in a light blue shirt speaking.

So therefore, what are side channels right these are typically covert channels which leak information which the designers or the cryptography algorithm did not consider and often you will find that they take place because of optimizations ok. These optimizations either are done intentionally by us or unintentionally by maybe the compiler or by other artifacts, which essentially tries to optimize our designs and try to only concentrate on performance ok.

For example, when you are trying to realize the square and you know an exponential algorithm which you want for example, for RSA the reason why you have square and multiply is because of efficiency right you have an if else structure. So, right you ensure that you get the result in a very efficient manner and that is precisely the thing which would target inside channels ok.

So therefore, the optimizations are often you know like what are targeted by a right side channel adversary. So therefore, our optimization should also take care of such threats if you really one an end to end security

(Refer Slide Time: 30:35)

The slide is titled "Cryptographic Implementations" and features a yellow background with a dark blue curved shape on the right side. At the bottom, there is a blue banner with the Swamyam logo and a video feed of a man in a light blue shirt. The text on the slide is as follows:

Cryptographic Implementations

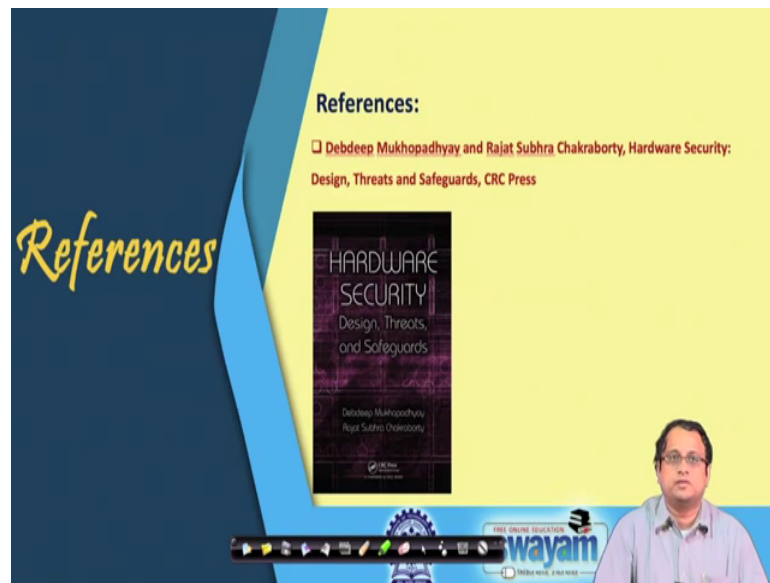
Cryptography is always an overhead.

- For security, the algorithms are very computation intensive.
 - symmetric algorithms have large number of rounds, uses non-linear Boolean functions.
 - asymmetric algorithms operates on large numbers, performs complex mathematical operations.
- Design Challenges: Performance, Size, Power.
- Further, need to tackle information leakage through covert channels: Secured Implementations.

So therefore, there are lot of challenges for a cryptographic designer, it has to be of course, take care of overhead as I said for cryptography the algorithms are very computationally intensive, if you are talking about symmetric key algorithms there are large number of rounds it uses non-linear Boolean functions, we needs to be suitably implemented asymmetric algorithms of an operator large numbers it performs complex mathematical operations so we need to make them efficiency.

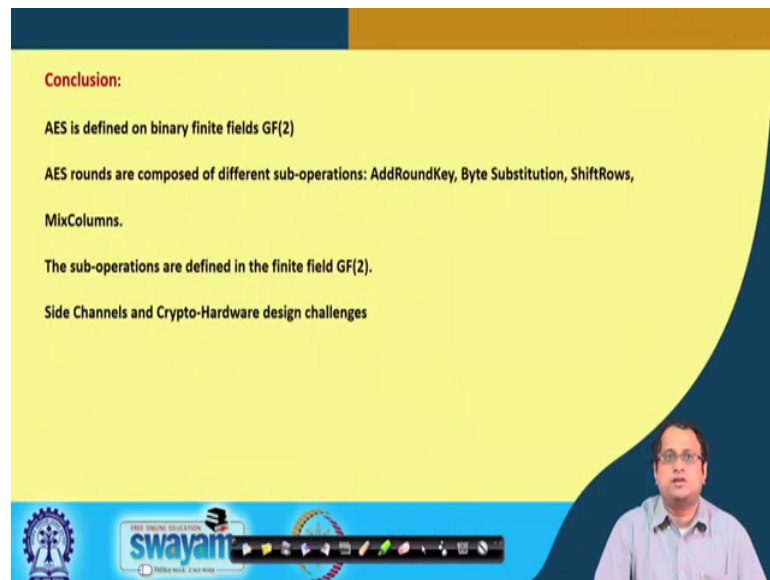
So therefore, they are definitely the usual challenges of performance size and power, but to make it more interesting you also have to tackle information leakages through go what channels like side channels and therefore if you really want a secured implementation you need to have an end to end spectrum of things.

(Refer Slide Time: 31:17)



So, again I would like to stop here and again this is my usual reference for the further for the for this part of the lecture.

(Refer Slide Time: 31:25)



And what we have essentially discussed today is that we have discussed about AES we have looked into the AES steps like Addroundkey Byte Substitution ShiftRows Mix columns and also the essentially discussed that these sub operations are defined in the finite field GF 2.

We also discussed about what are side channels and we kind of reflected that if you really want to have an end to end cryptographic hardware, then you also need to look into not only performance but also these threats and that essentially we will see in the following classes of how to slowly address on. We will try to understand not only the attacks but also how to counter them in the future classes ok, so at this I would like to thank you for your time.