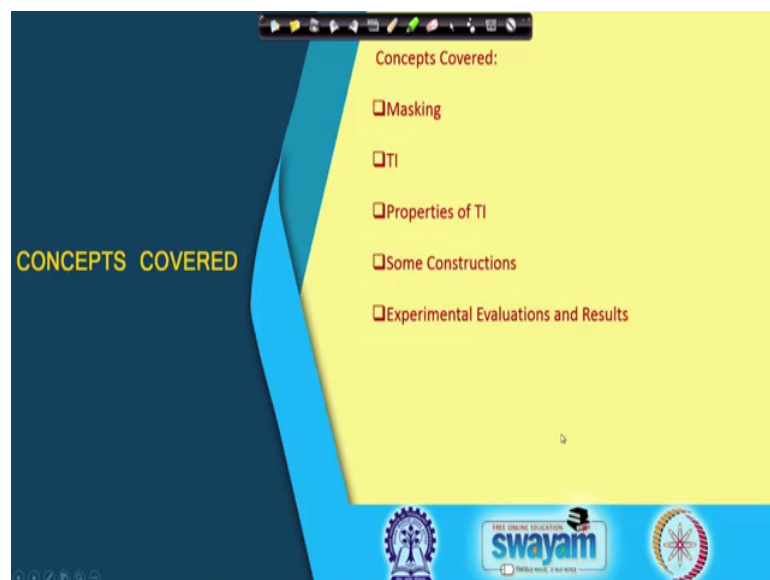


Hardware Security
Prof. Debdeep Mukhopadhyay
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 43
Power Analysis Countermeasures

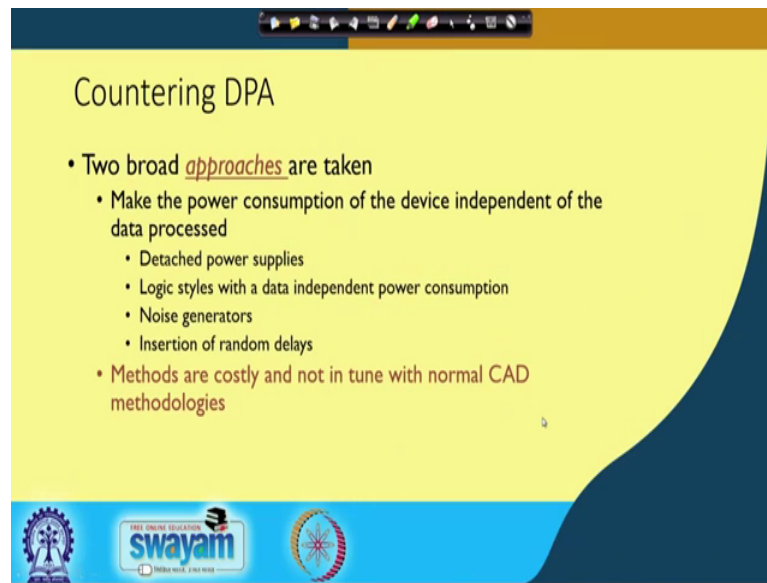
Welcome to this class on Hardware Security. So, we shall in the last few classes we have been discussing about power analysis in today's class, we shall be trying to see how we can Counter Power Analysis.

(Refer Slide Time: 00:27)



Specifically we shall be talking about a technique which is called as masking, which is a very popular strategy to prevent or protect against power attacks and then we shall be discussing about some pitfalls of masking and that will lead us to the introduction of a technique which is called as TI or which is an abbreviation of threshold implementation. We shall be trying to look into some of the properties of TI and also see some possible constructions and we will finally conclude with some experimental evaluations and results on a specific case study.

(Refer Slide Time: 00:59)



The slide is titled "Countering DPA" and features a yellow background with a dark blue curved shape on the right side. At the top, there is a navigation bar with various icons. The main content is a bulleted list:

- Two broad approaches are taken
 - Make the power consumption of the device independent of the data processed
 - Detached power supplies
 - Logic styles with a data independent power consumption
 - Noise generators
 - Insertion of random delays
 - Methods are costly and not in tune with normal CAD methodologies

At the bottom of the slide, there are three logos: the IIT Bombay logo on the left, the Swamyam logo in the center (with the text "FREE ONLINE EDUCATION swamyam" and "IIT BOMBAY, PUNE, KOLKATA"), and another circular logo on the right.

So to start with, how we can counter DPA? So, we as we have seen write DPA or differential power attacks, fundamentally works by exploiting the fact that power consumption depends upon the underlying data. So, these so there could be several strategies and the strategies can be broadly classified into two approaches. The first approach which is essentially a very popular is essentially by trying AdHoc design techniques which essentially with objective of making power consumption of the device independent of the underlying data.

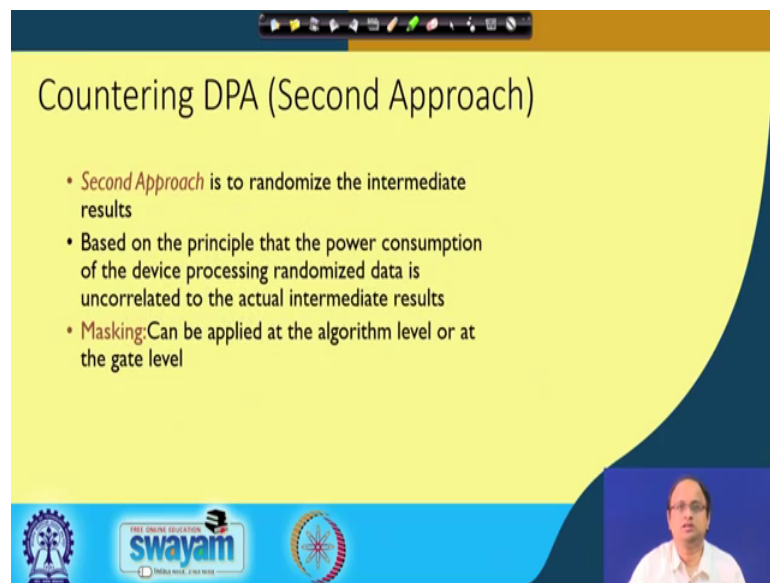
For example right you can actually try to have a detached power supply you can try to use logic styles with data independent power consumption for example, you can try to bring in complementary logic for example, in one case right if you are complementing with x in another circuit you are you are operating on \bar{x} . So, the idea is that if there is some net which is making a 0 to 1 toggle, there is some other net which is making a 1 to 0 toggle. Likewise right, you can also try to bring in techniques like noise generators, inserting random delays, but the point is right most of these techniques are not essentially in tune with the normal CAD methodologies and also right the methods are often costly

More importantly, these techniques being AdHoc do not guarantee against protections against these kind of attacks. So, therefore, right it may happen that for example, if you are talking about a complementary logic although in principle it looks very nice because it would try to kind of pretty much make the consumption independent of data by making

it constant, but then it would also depend upon how you are routing your design inside an (Refer Time: 02:40). For example, if the two circuits like the one which is processing on x and the circuit which is processing on \bar{x} are not routed in a very uniform manner, then it may happen to still or it may still need to you know like I would say like it may still need to non uniform power consumption which would still be dependent upon the underlying data.

So, therefore, right these techniques often are costly and at the same time and you know like do not give us an end to end security, but more importantly right these methods are not amenable to our CAD methodologies and therefore, right the designer in the very first place cannot try to develop or cannot develop his or her design with suitable countermeasures.

(Refer Slide Time: 03:21)



The slide is titled "Countering DPA (Second Approach)" and features a yellow background with a dark blue curved border on the right side. At the top, there is a navigation bar with various icons. The main content consists of three bullet points:

- *Second Approach* is to randomize the intermediate results
- Based on the principle that the power consumption of the device processing randomized data is uncorrelated to the actual intermediate results
- *Masking*: Can be applied at the algorithm level or at the gate level

At the bottom of the slide, there are three logos: the Indian Institute of Technology (IIT) logo on the left, the "swayam" logo in the center, and another circular logo on the right. A small video inset in the bottom right corner shows a man with glasses speaking.

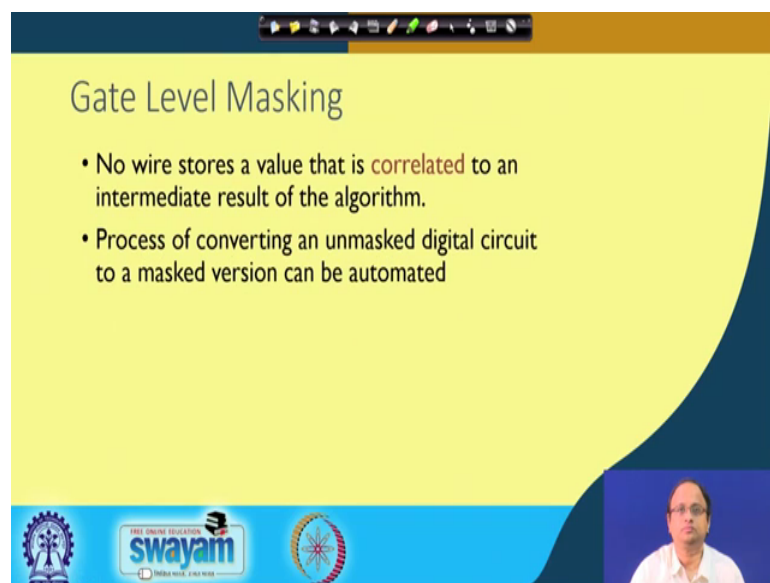
So, this leads us to a second approach, which is essentially again with the objective to randomize intermediate results, but in this case right you can actually apply this at the algorithm level or at the gate level. So, therefore, pretty much you can implement them and for example, the RTL level of the Verilog level and therefore, these methods are more conducive and more desirable from the design point of view.

So, therefore, these are again you know like based on the principle that power consumption of the device processing randomized data is uncorrelated to the actual intermediate result. So, therefore, the idea is we will see is right it basically is built upon

this idea that if we can randomize the internal information and essentially process or when we are and do our computations on the randomized information, then the power essentially will be dependent upon the randomized data and not on the actual intermediate results. So, therefore, right it would be statistically independent of the actual secret which is being processed

So, masking is one such technique and it is a very popular technique which has been adopted widely for protecting against power attacks.

(Refer Slide Time: 04:28)



The slide is titled "Gate Level Masking" and features two bullet points. The first bullet point states: "No wire stores a value that is correlated to an intermediate result of the algorithm." The second bullet point states: "Process of converting an unmasked digital circuit to a masked version can be automated". At the bottom of the slide, there are three logos: the Swayam logo, the logo of the Ministry of Education, Government of India, and the logo of the National Institute of Technology (NIT) Patna. A small video inset of a speaker is visible in the bottom right corner of the slide.

So, principle of masking essentially is based upon the fact that no wire stores a value that is correlated to an intermediate result of the algorithm and the process of converting an unmasked digital circuit to a masked version can also be automated and therefore, right this method is very popular and desirable from the CAD perspective.

So, as we will see right, in this kind of circuits there are two distinct parts, the one part which basically processes on the mask and the second part right which processes on the mask data. So, the idea is that if you are processing on a data right for example, any intermediate value you typically try to kind of split it into two parts; one part which is the mask which is essentially hiding your information and the other part which is the mask data which is essentially nothing, but data camouflaged with the mask.

So, we have to basically process both of them in independent fashion and we have to keep in very careful that no intermediate circuit is combining these two parts because if we combine the mask along with the mask data then the unmask data is essentially exposed and therefore, right we have to do the computation in a manner so that the actual data does not get opened up during the computation.

(Refer Slide Time: 05:42)

The slide titled "Masked AND Gate" contains the following content:

- Block Diagram:** A box labeled "Masked Gate" has four inputs: a_m , m_a , b_m , and m_b . It has one output: q_m .
- Equations:**
 - $a_m = a \oplus m_a$ (with handwritten note $a, b \times$)
 - $b_m = b \oplus m_b$ (with handwritten note a_m, b_m, m_a, m_b)
 - $q_m = q \oplus m_q$ (circled in red)
 - $q = f(a, b)$ (with handwritten note "different mask")
 - $q_m = f(a_m, m_a, b_m, m_b, m_q)$ (with m_q circled in red)

So, let us start with a very simple example of trying to mask a fundamental gate. So, this mask this is nothing but the AND gate and we would like to mask the AND gate ok. So, as we know that in the AND gate, there are two important two inputs like it is an input of a and b and you basically give an output which is say y or q. So, q is essentially a function of a and b where a and b are your actual inputs it could be like single bit values.

So, in an in an normal unmask data we would basically do a processing on a and b and therefore, the power consumption since depends upon a and b would leak information about a and b and that is what we have seen when we are studying about d about DPA and various versions of that. In a masked counterpart of this order in a masked AND gate what we will do is that, we will basically take a which is a actual data, but rather than processing on a we will basically mask a with a random value which is m a ok. So, m a is my mask.

So, now once I once I for example, you know like so one way of masking is by applying a bitwise XOR. So, we take a and we XOR it with m a, where m a is randomly our value

for either 0 or 1. So, it is again a 0 1 random value when I XOR it I get a m. So, a m becomes my masked value and m a is called as the mask.

So, essentially right I mean you have got two parts. So, you have got for example, this part which is essentially your mask, which is essentially randomly chosen so it is probably randomly chosen from a 0 or 1 value, on the other hand I this part is essentially what we call as the masked data ok. So, this is the masked data. So, now, once you basically create these mask likewise you also mask b by using the mask m b and creating the mask value b m.

Now, your circuit should process on a m ok. So, it is it should process on a m one second it should process on these two parts, it should process on a m, b m, m a and m b. So, in no case right it should process on a and b. So, the idea is that a and b should never be processed together.

So, so therefore, right we will basically kind of convert this function f into another function say f hat where f hat basically processes on these data these masked values and you can see I have also used another mask which is essentially what is called as the output mask ok. So, the idea is that if you take these output mask; if you take this output mask and the circuit produces a q m, then the final result is obtained by using this equation which is nothing, but q essentially is equal to the XOR of q m and m q.

But note that in any case right, if you have got a following circuit, you will basically suppose the following circuit operates on q, but then the subsequent circuit also should be processing on q m and m q. If I basically take q m and I combine with m q, then again this data gets exposed which is not desirable so; that means, right in a subsequent circuit I should again have a masked gate which will again process on m q and q m, but we would not combine them. So, therefore, in no case right the masked data and the masked value or the mask should not combine ok. So, therefore, the entire thing should be processed in an independent manner through independent circuits.

So, let us now see for example, how we can mask or how we can convert the AND gate into its masked counterpart in a way like in another way words right we basically would like to see how to write f hat right given or know knowing that f is nothing but the AND of a and b. So, that is what we will basically see subsequently.

(Refer Slide Time: 09:33)

Masked AND Gate

$$\begin{aligned}q_m &= (a \cdot b) \oplus m_q \\ &= (a_m \oplus m_a) \cdot (b_m \oplus m_b) \oplus m_q \\ &= ((a_m \cdot b_m \oplus b_m \cdot m_a) \oplus (m_b \cdot a_m)) \oplus m_a \cdot m_b \oplus m_q\end{aligned}$$

b_m (a_m \oplus m_a)

swamyam

So, so this brings us to this mask computation. So, basically right in a way what we want to do is, we want to calculate a into b, but remember that there is a output mask say m k m q. So, we know that a is nothing, but a m XOR with m a and b is nothing but b m XOR with m b and XOR with m q.

So, therefore, right if I break it up then this is essentially pretty much the computation which is done ok. So, you can see that it is nothing but, we have basically applied the distributive property and basically kind of spread out the computations in this manner. So, note that in this computation, there is no way right essentially we would basically what we are trying to do is basically we are trying to write the computation so that actual data a and b are not coming into the into play. At the same time, we have to also you know like slightly restructure this computation because if I implement the circuit as it is as shown here in this particular equation, then it will again lead to a leakage for example, you can observe that if I you know like if I do the computation as it is like this then this is nothing, but b m XOR with m a, a m XOR with ma and this data is nothing but a.

So, therefore, right although from the associativity properties, this is essentially I would be you know like I mean this is this is so this is correct, but from the security leakage point of view right this essentially is not the correct way of doing the computation.

(Refer Slide Time: 11:23).

The slide is titled "Masked AND Gate" and features a yellow background with a dark blue curved border on the right side. At the top, there is a navigation bar with various icons. The main content consists of four bullet points:

- There are $4^5=1024$ possible input transmissions that can occur.
- It turns out that the expected value of the energy required for the processing of $q=0$ and $q=1$ are identical.
- Thus protected against DPA, under the assumption that the CMOS gates *switch only once in one clock cycles*.
- But we know there are glitches, and so the output of gates swing a number of times before reaching a steady state. Hence... the argument continues.

At the bottom of the slide, there is a blue banner with logos for "swayam" and "INDIA WISE, LEAD WISE". A small video inset in the bottom right corner shows a man with glasses speaking.

So, basically kind of we have to rewrite or we have to kind of implement them in a way so that in no case right we have got these kind of situations arising that is the mask data and the mask should not combine together and this leads us to a specific circuit which I will explain after this and basically right I mean the whole objective of doing this masking is that as we know that there are 4 inputs I mean we know that we actually know that there are 5 inputs taking the output mask also into account and each of them have got you know like 4 possible transitions like 0 to 0, 0 to 1, 1 to 0 and 1 to 1. So, note there are totally 1024 possible input transitions that can occur.

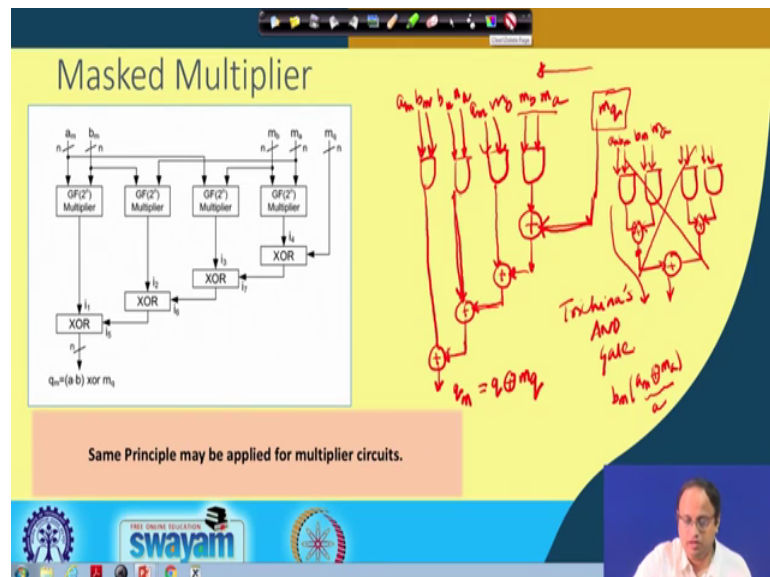
So, if you remember like we previously in one of our classes, we discussed about why AND gates leaked ok. So, where there we saw that the average energy for e of q equal to 0 and e q equal to 1 are not equal. So, if you do the same exercise for the masked gate, you can essentially do it again considering that the masked gate is a 5 input Boolean function you can observe that it will turn out that the expected value of the energy required for processing of q equal to 0 and q equal to 1 will be identical now, showing us that if the gate essentially performs one transition at one time then there is no leakage.

However and that essentially means these are protected against DPA under the assumption that the CMOS gates switch only once in one clock cycle, but we will come to a you know like a different argument so, very soon where we will see that if there are something which are called as glitches where essentially you know the output of the gate

will swing a multiple number of times before reading reaching a steady state, then still there can be a leakage and the design right even after this protection can still be vulnerable against power attacks.

So, at this point of time, let us consider that there are no glitch and let us assume that the gate is switching only once in one clock cycle. Under this assumption, right we can prove that e of q equal to 0 and q equal to 1 will be identical exactly in the same way we prove that it is not identical in the case of a normal AND gate ok, proving that this circuit essentially should be secured against first order DP attacks.

(Refer Slide Time: 13:23)



So, so therefore, right as I said that; however, if I want to really achieve that, we have implement a circuit in a slightly clever way so that the masked data and the masked value I mean the masked value and the mask do not combine together. So, one popular masked implementation is shown in this diagram, this is also called as stretchiness gate I mean if you for example, so this is essentially a masked multiplier which has been shown, but in a similar way you can also think of a masked AND gate for example, what you can think of is you can replace these multipliers by AND gates for example you can think that pretty much there are like 4 AND gates ok. So, these are the 4 AND gates which are together and there is an XOR. So, we can basically start processing on the inputs like $m \cdot b$ and $m \cdot a$. So, these are normal AND gates and you can take the output mask which is $m \cdot q$ and you can basically combine it with an XOR ok, so this is your

XOR. So, therefore, this is your output mask $m \oplus q$ which is coming into exhorting the output of $m \oplus a$ into $m \oplus b$ and then again you take these two inputs which are essentially nothing, but $m \oplus b$ again ok, so this is $m \oplus b$ and this is $a \oplus m$ and likewise let us complete this part. So, you are you have got $b \oplus m$ and this part is $m \oplus a$ and likewise these two parts are $a \oplus m$ and $b \oplus m$ right.

So, you basically take this two part and you again combine this by an XOR. So, there is an XOR here, which is written over here you take these two this part. So, you can basically combine this by an XOR and again you take an XOR function and this is your finally, this part right is your $q \oplus m$ because $q \oplus m$ means because $q \oplus m$ is nothing, but the XOR of q and $m \oplus q$ ok. So, this gate is also called as Trichinas AND gate. So, you can observe that the whole; the whole way right the; I mean the principle of doing this computation is to bring in this $m \oplus q$ which is completely unrelated to a and b at the very beginning ok.

So, you basically take two random values like $m \oplus b$, $m \oplus a$ and you XOR it with $m \oplus q$ right and you start the computation from this side actually. So, therefore, right once you have done this computation, you basically have got $a \oplus m$, $a \oplus m \oplus b$ and then you are basically combining them with this part and then you get this result you basically combine with this XOR and then finally, you get this result. So one should be you know one can be tempted so basically you can say this is a very skewed circuit. So, a possible temptation could be that I would like to you know like make this circuit is little balanced and if you do that right you will always see that you can end up in combining the masked data and the masked value and therefore, right it is really a good exercise to see right that what happens if you try to balance this circuit and it will happen that if you try to balance this circuit then you are combining say a term like you know like for example, if I consider a balanced circuit ok, so one may you know like by applying the principles of digital logic right for example, can end up doing this computation. So, you know like I may be tempted to say you know like combine these two parts apply an XOR here and again combine these two parts apply an XOR here and then I would like to apply an XOR here ok.

So, you can see that this basically makes the circuit more balanced and also probably reduces the critical delay of the circuit, but if you do that right then what you are doing if you observe for example, this point and if you observe this computation say this is $a \oplus m$,

this is b m and this is b m and this is m a ok. So, again right this part is nothing, but if you take b m common it is the XOR of a m and m a and this part is again where you are exposing a ok.

So, therefore, right this is a; this is a wrong approach ok. So, we cannot actually balance the circuit in this way and therefore, right finally, I have got a circuit which is unbalanced, but at the same time right is essentially secured against first order leakage if there is no glitch in the circuit.

So, you can apply the same principle for realizing a multiplier where you basically replace these AND gates by corresponding multipliers. So, you can observe that, you know that the cost of getting this security is significant because in order to realize 1 AND gate, now you have got 4 AND gates and you have got 4 XORs ok. So, therefore, there is a significant blow up in terms of chip area and other peripheral costs, but at the same time right is this is essentially sound in terms of DPA leakage.

So, now we will see you know like that whether that's the end of the story or there are more things to be considered.

(Refer Slide Time: 18:46)

Masking and 1st order Analysis

- In these masking designs, the intermediate variable X is split into two random variables X_1 and X_2 , st. $X_1 \oplus X_2 = X$.
- Assume the leakage $L(X) = HW(X_1, X_2)$, we have the following:

x	x_1	x_2	$L(X)$	Mean($L(X)$)	Var($L(X)$)
0	0	0	0	1	1
0	1	1	2		
1	0	1	1	1	0
1	1	0	1		

Masking does not reveal any information from 1st-order analysis, as the mean is constant for different values of x . However, a 2nd-order analysis can reveal because of the dependence of variance on x .

So, so therefore, right I mean we will basically consider masking and first order analysis and here is a very an interesting way of observing or taking a relook at masking. So, in this masking design, what we have done basically is that the intermediate variable X is

split into two random variables X_1 and X_2 , such that $X_1 \oplus X_2$ is equal to X and assume that again if I you know assume that the leakage is kind of modelled by the hamming weight of X_1 and X_2 , note that now you are not storing x in an register, but the registers are storing X_1 and X_2 like the shares of X . So, X has been broken up into X_1 and X_2 , like we have broken up a into a_m and a_{m-a} so there are two parts that you have broken up them broken them up into and therefore, the power consumption will essentially depend upon the hamming weight of both X_1 and X_2 .

So, if you just make a simplistic model like that and try to observe how essentially is the leakage. So, you can see that, here this diagram shows or this table shows that my input is say X which is 0 or 1, if I break it up if I write 0 right, then I would I can write the masks as either 0 0 or 1 1 because both of them XOR to 0 and likewise right I can write 1 as 0 1 or 1 0.

So, now observe what is the leakage because of this x ok. The leakage here is 0 because the leakage was 0 0 so that is 0, the leakage for 1 and 1 should be maximum that is 2 ok, the leakage here is 0 1 which is 1 and the leakage here is 1 0 which is 1. So, now, if you observe the mean leakage, the mean leakage right essentially so mean leakage means if I take the mean when the input is say 0 ok, the values here are you know like 0 and 2. So, if I take 0 and 2, the mean is 1, likewise right here if I take for the mean for you know that the leakage for when the value is 1, it is here 1 and 1 so the mean is still 1.

So, therefore, right from the first order point of view like when we are just like doing first order when we are applying first order statistics that is when you are computing the mean and doing your you know like DOM or correlation power attacks as we have seen, the mean is not leaking any information and that is why right this is secured against first order you know like or potentially secured against first order attacks, but if you observe the observe the variance you will see that here the variance is significantly larger compared to here, the variance here is 0 where the variance is here is 1 and therefore, right from the if I try to do a second order analysis this will still reveal and you know like because of this dependence of variance on x . So, this is important to keep in mind that masking right in as we have defined is essentially first order resistant, but not resistant against the second order analysis and therefore, right we can still we still like to improve the basic design if you want to protect against a higher order attack.

So so, therefore, right I mean this is so therefore, the whole idea is that the so let us see how we can develop basic the basic scheme, but before we go into that right what we will also try to see is what happens you know like so, let us first of all define what is a higher order masking.

(Refer Slide Time: 21:54)

Higher Order Masking

- Thus in a d th order masking aims to randomize intermediate sensitive data X by splitting into $d+1$ uniformly distributed variables X_1, \dots, X_d, X_{d+1} , st:

$$X = X_1 \perp X_2 \perp \dots \perp X_d \perp X_{d+1}$$
- Depending on the exact \perp operator, we can have multiplicative, additive masking.
- When the \perp operator is XORing, we call it Boolean Masking.
- Each variable X_i is referred to as a secret share and the secret sharing can be done by randomly generating X_1, X_2, \dots, X_d , and calculating X_{d+1} .

So, in a so therefore, right we as we see that our design is not secured against second order attacks. So, likewise right it is of course, not secured against a third order attack, fourth order attack and so on. So, therefore, in if I generalize this in a d th order masking what we will try to do is we will aim to randomize the intermediate sensitive data X by splitting into d plus one uniformly distributed variables. So, like when we are protecting against first order attacks, we have broken up our input x into 2 parts ok. If I want to protect against a second order attack, we have to break it up into 3 parts; X_1, X_2 and X_3 such that they XOR up to get me X .

So, in a general generic setting right if I have got say X_1, X_2, X_3 and so on till X_d and X_{d+1} and I use some kind of operator to indicate how we are basically combine them note that the masking right essentially the way we have seen is that this operator is XOR and this particular way of doing masking is called as Boolean masking. As opposed to this, there is another alternative way of doing masking where I apply for this operator I apply a plus that is the integer addition and that is called as additive masking.

So, each variable X_i , so in this case so you can apply both Boolean masking as well as additive masking and each and you can observe the way in which we obtain the shares is that if I want $d+1$ shares, I will randomly choose X_1 to X_d and then I will choose or calculate X_{d+1} , I will compute X_{d+1} so that this equation is satisfied and that you can easily verify that for example, if this is XOR what I do is I I kind of randomly choose X_1 to X_d and then I basically compute X_{d+1} so that their XOR is equal to X which you can easily get by XORing X with the XOR of X_1 to X_d .

So, therefore, at each variable X_i is referred to as secret share and a secret sharing can be done by randomly generating X_1 to X_d and by computing the value of X_{d+1} ok.

(Refer Slide Time: 24:03)

Hiding within the Mask

- Given an input sharing, all the cipher operations are done inside a mask:
 - Linear Transformations
 - S-Box computations
- Linear transformations are easy:
 - Thus, $l(X) = l(X_1 \oplus X_2 \oplus \dots \oplus X_{d+1}) = l(X_1) \oplus \dots \oplus l(X_{d+1})$
- So, we can perform the linear operations on the masks.

So, so now we would like to see that how we can really hide behind the mask. So, given an input shared there I mean I mean see how we can apply masking. So, 2 ciphers. So, note that the ciphers can be typically having 2 types of transformations it can have linear transformations or it can have non-linear transformations ok, for linear transformations actually masking is very easy. So, because of this simple fact that if I break up X into the shares like X_1 to X_{d+1} then I know that by because of the fact that the linear l is linear, when I apply l of on X_1 XOR so on till X_{d+1} that is equal to l of X_1 XOR of l of X_2 XOR of l of X_{d+1} .

So, therefore, right we can perform the linear operations on the masks ok. So, you can still actually break, when the moment you have broken up the input X into parts you can

apply the linear transformations on each of these parts separately and you know that when you combine these results you are getting the correct output ok. So, there is no problem in that case.

(Refer Slide Time: 25:04)

Nonlinear Masking

- It is challenging for nonlinear functions.
- Example: $f(X, Y) = Z \oplus XY$
- Masked Circuit:
 - $f_1(X_1, Y_1) = Z_1 \oplus X_1Y_1$
 - $f_2(X_1, X_2, Y_1, Y_2) = ((Z_2 \oplus X_1Y_2) \oplus X_2Y_1) \oplus X_2Y_2$
- Note again the ordering of the operations is very important!
 - Don't do, $f_2(X_1, X_2, Y_1, Y_2) = (Z_2 \oplus X_1Y_2) \oplus (X_2Y_1 \oplus X_2Y_2)$...as the second parenthesis is dependent on Y
- However, this is not secure against higher order attacks.
- Actually, not even 1st order attacks if there are glitches.

So, likewise but you know like when you apply the process for a non-linear transformation, then the process is much more complex. For example, right when I want to compute say $Z \oplus X \oplus Y$, where $X \oplus Y$ as we know is a non-linear combination. So, therefore, right one way of possibly, potentially masking this is as shown here. So, I basically do $Z \oplus X \oplus Y$ and in the other share I basically calculate $Z \oplus X \oplus Y \oplus X \oplus Y$.

So, note that you know like in this case right, when I combine f_1 and f_2 then because of the combination of $Z \oplus X$ and $Z \oplus Y$ I get Z and $X \oplus Y \oplus X \oplus Y$ if I combine right I get $X \oplus Y$ into $Y \oplus Y$ which is equal to Y XOR of again $X \oplus Y$ if I take common $Y \oplus Y$ will be equal to; will be equal to Y . So, therefore, right if I take Y common then I get $X \oplus Y$ XOR Y which is equal to X therefore, I get $X \oplus Y$. So, therefore, this is correct, but at the same time you should observe that the you should observe the parentheses here, this is very important because if this ordering is not properly done then again $X \oplus Y \oplus X \oplus Y$ will combine with $X \oplus Y$ which will lead to the leakage of the information on Y and therefore, right I mean one should be very careful when you are applying you know like; you know like associativity in this particular cases.

So, for example, here what we will do is, we will kind of combine $X_1 Y_2$ with an independent data Z_2 and then I will combine it with $X_2 Y_1$ and then I will combine the result with $X_2 Y_1$ which is essentially you know trying to keep it in secured against a first order attack, but as we know that this is not secured against second order attack because the moment I combine the leakage of f_1 and f_2 , then all my shares are leaked and therefore, the actual data is also compromised. Actually this is not even first order attacks which is where the unfortunate news ok, this is essentially not secured if there are something which are called as glitches inside a circuit ok. So, that essentially we will study up you know like in the next class, where we will see right what is the effect of glitches on these kind of circuits.