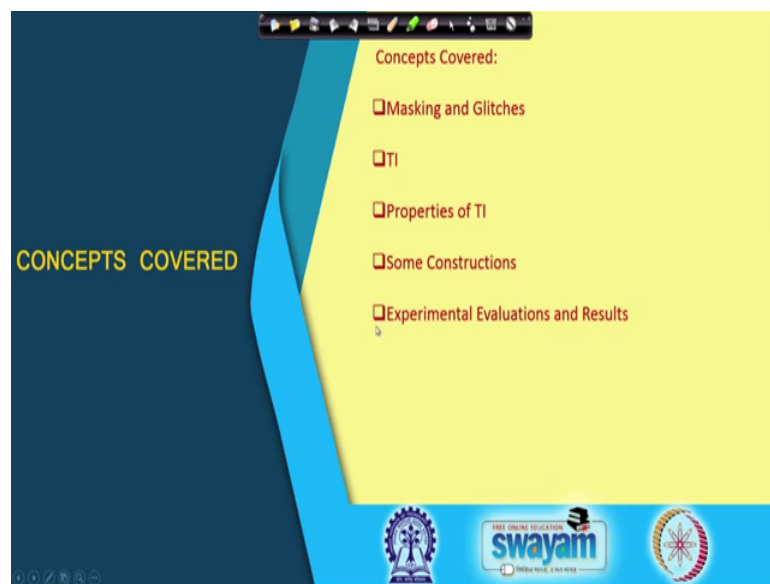


Hardware Security
Prof. Debdeep Mukhopadhyay
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 44
Power Analysis Countermeasures (Contd.)

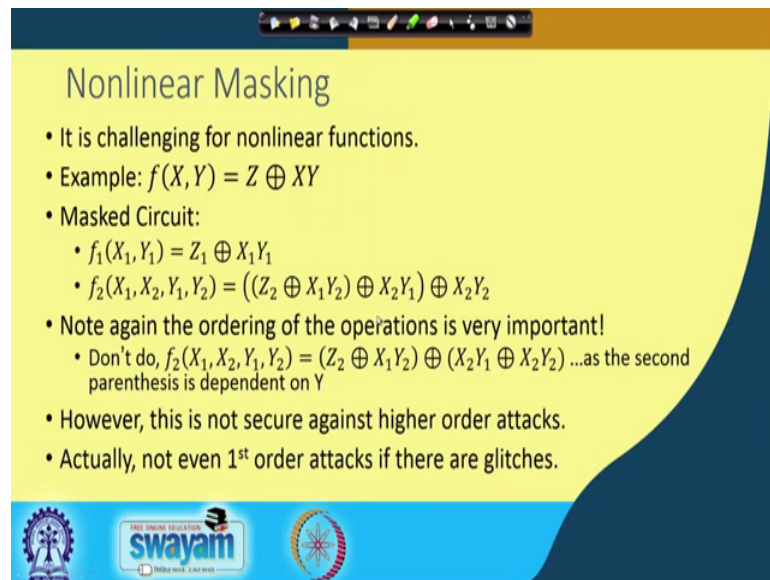
So, welcome back to this class on Hardware Security. So, we shall be continuing our discussions on masking.

(Refer Slide Time: 00:23)



So, in the last class we essentially defined how masking works, but in today's class we shall be trying to see how masking and glitches essentially work together in a sense like how glitches can be applied to break masking.

(Refer Slide Time: 00:36)



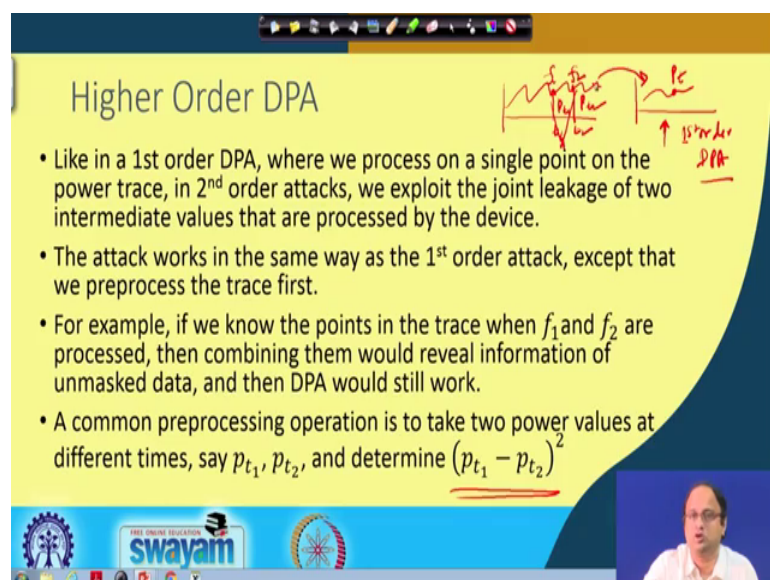
Nonlinear Masking

- 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.

Logos: IIT Bombay, swayam, IIT Madras

So, we shall be, so, let us continue our discussions on this topic. So, as we have seen in the last class right, if we want to mask a non linear circuit like Z XOR XY, one potential way of masking is shown here and theoretically they should be secured against a first order attack. But, we discussed right this is not even secured or not secured in against first order attacks if there are glitches.

(Refer Slide Time: 00:58)



Higher Order DPA

Handwritten diagram showing two points on a trace, labeled p_{t1} and p_{t2}, with an arrow pointing to the text '2nd order DPA'.

- Like in a 1st order DPA, where we process on a single point on the power trace, in 2nd order attacks, we exploit the joint leakage of two intermediate values that are processed by the device.
- The attack works in the same way as the 1st order attack, except that we preprocess the trace first.
- For example, if we know the points in the trace when f_1 and f_2 are processed, then combining them would reveal information of unmasked data, and then DPA would still work.
- A common preprocessing operation is to take two power values at different times, say p_{t_1}, p_{t_2} , and determine $(p_{t_1} - p_{t_2})^2$

Video inset: A man speaking.

Logos: IIT Bombay, swayam, IIT Madras

So, in order to understand that right we will try to take a little bit look into what essentially; so, before I go into that right we will basically take a little bit of look into

what are Higher Order DPA. So, in higher order DPA this is exactly like a first order DPA where in a first order DPA as we have seen right where we process on a single point on the power trace . In second order attacks we exploit the joint leakages of two important or two intermediate values that are being processed by the device.

So, basically it essentially means that if there is a power trace and suppose right here basically you know like this is your corresponding power trace and in the previous case when you are doing your first order attack you are basically targeting on one point on the power trace. But, if you are doing a second order attack then you are basically targeting two points on the power trace, ok. It may be happen that, it may happen that here you are processing f_1 and here you are processing f_2 .

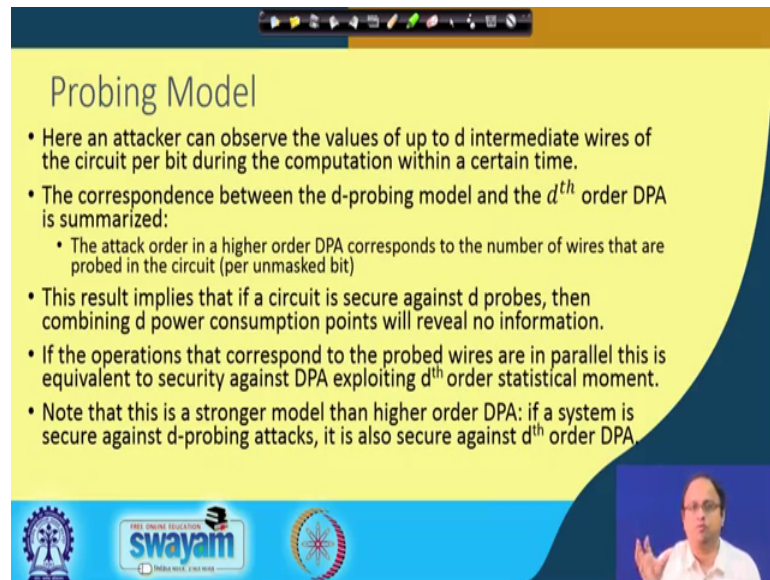
So, now we will be considering the joint leakage of f_1 and f_2 and we and as we have seen right that if f_1 and f_2 , right together depends on all the shares of the input of the inputs of the circuit, then essentially it will still leak the information about the actual data, ok. So, therefore, right we here exploit the joint leakage of two intermediate values that are processed by the device that are works in the same way as the first order attack except that we pre process the trace first.

So, for example, right if we know that the points in that trace where f_1 and f_2 are processed then I would try to combine them their leakages, and then right I will basically kind of kind of preprocess the trace and create another preprocessed trace and then I will be applying a first order DPA on that. So, what I will be doing is suppose you know like I observed the power consumption here and the power consumption here right at this point is say p_{t_1} and the power consumption here is p_{t_2} because these are two time instances t_1 and t_2 . So, then I will basically combine for example, these value like $p_{t_1} - p_{t_2}$ whole square.

So, therefore, basically I will now replace these two power traces by a single point and another power trace which is say p_t , which is essentially the nothing, but the you know the difference and then taking the square you can of course, replace it by some other function, but this is a very common preprocessing operation. And, then once you have preprocessed this power trace then you basically apply a normal first order DPA as we have seen, ok. So, like a normal correlation power attack, for example.

So, therefore, the whole idea is that you are trying to kind of combine the leakage of two points, and that essentially as we have seen is that if it opens up the shares then basically the countermeasure is not secured against such kind of attacks.

(Refer Slide Time: 03:42)



Probing Model

- Here an attacker can observe the values of up to d intermediate wires of the circuit per bit during the computation within a certain time.
- The correspondence between the d -probing model and the d^{th} order DPA is summarized:
 - The attack order in a higher order DPA corresponds to the number of wires that are probed in the circuit (per unmasked bit)
- This result implies that if a circuit is secure against d probes, then combining d power consumption points will reveal no information.
- If the operations that correspond to the probed wires are in parallel this is equivalent to security against DPA exploiting d^{th} order statistical moment.
- Note that this is a stronger model than higher order DPA: if a system is secure against d -probing attacks, it is also secure against d^{th} order DPA.

The slide also features a small video inset of a speaker in the bottom right corner and logos for 'swayam' and other institutions at the bottom.

So, so, therefore, right I mean so, therefore, this brings us to a model which is essentially also used hand in hand with our d^{th} order attack which is called as Probing Model, and this is a very useful model to analyze what are glitches or the glitches inside the circuit. So, idea is that here the attacker can observe the values of up to d intermediate wires of the circuit per bit during the computation within a certain time.

So, these are more stronger form of adversary where we are trying to kind of allow the attacker to observe the values of up to d intermediate wires. So, suppose you have got a circuit. So, remember for example, the mask circuit and then align the attacker to observe say your two intermediate wires or maybe three intermediate wires depending upon the you know like my notion or model of the adversary.

So, in the general case suppose the attacker observe d intermediate wires of the circuit per bit during the computation within the certain time and the you know like and this model of the d -probing model is very much linked with what is called as what we have seen as the d^{th} order DPA, ok. And, the similarities because the fact that the attack order in higher order DPA corresponds to the number of wires that are probed inside the circuit per unmasked bit.

So, because right when you are you know like so, the idea about the higher order DPA would be to basically kind of combine multiple leakage points and in the probing model also you are trying to probe inside the circuit and you are basically observing the information of those in the of those observed points. So, you are allowing the attacker now to observe say d points inside the circuit, and this is essentially kind of similar to the higher order DPA were essentially here and essentially has got a very close relation between them.

So, this relation or these result implies that if your circuit is secured against d probes then combining d power consumption points will know will not reveal any information and therefore, right it should be secured against a d th order adversary. So, if the operation so, you know if the operations that correspond to the probed wires are in parallel like; suppose, you know like if you are observing multiple points together and they are working all together their working simultaneously, then this equivalent to the security notion that we have against d th order d th order d th order DPA; that means, where you are doing a DPA with d th order statistical movement.

Like as we have seen in a second order attack you do an attack with a second order statistical movement, ok, and not like the first order statistical movement which we are usually doing in our normal first order DPA attacks. So, note that this is a stronger form of model; model probably stronger the higher order DPA. If a system is secured against d probing attacks then you should be also secured against the d th order DPA.

And, therefore right we were basically trying to, try to use these probing model to analyze the security of a given cipher. So, idea is that if you can observe, if you can ensure that your circuit is secured even when the adversary is able to observe d intermediate points then; that means, right even if it combines d , you know I mean d points in the power profile then also right our system should be secure, ok. So, that is essentially the in the inherent idea.

(Refer Slide Time: 06:53)

Security on a Glitchy Circuit

The probing model captures the effect of glitches by letting the attacker gather information on inputs, like Y_1, Y_2 , and intermediate values, X_2Y_1, X_2Y_2 , etc.

y	y_1	y_2	x_2	$x_2 \oplus x_1 y_2$	AND	XOR
0	0	0	0	0	0+0	0+0
0	1	1	0	0	0+0	0+0
0	0	0	1	0	0+0	0+0
0	1	1	1	0	0+0	0+1
0	0	0	0	1	0+0	2+0
0	1	1	0	1	0+0	2+0
0	0	0	1	1	0+0	2+0
0	1	1	1	1	0+0	2+1
1	1	0	0	0	0+0	0+0
1	1	0	0	0	0+0	0+0
1	0	1	1	0	0+0	0+1
1	1	0	1	0	0+0	0+2
1	0	1	0	1	0+0	2+0
1	1	0	0	1	0+0	2+0
1	0	1	1	1	0+0	2+1
1	1	0	1	1	0+0	2+2

Average glitch power for the AND gate does not depend on y .

Average glitch power for the XOR gate depends on y .

So, now we will basically is kind of analyze our mask gate, and we will see the mask gate and will we analyze the mask gate, align the observer to observe intermediate points. So, what we will do is in order to analyze this so, what we will try to do is basically we will try to kind of consider this truth table. So, in the truth table you can see that this is really nothing, but pretty much the masking that we have just now seen, ok. So, this is the masking that we are seeing when we are considering the functions f_2 .

So, if you remember right this is the function that we have seen here which is essentially $Z_2 \text{ XOR } X_1 Y_2$ and so, $\text{XOR with } X_2 Y_1 \text{ XOR with } X_2 Y_2$. So, here we are basically trying to observe this particular circuit, but under the presence of glitches.

So, the probing model actually, it is the important point to keep in mind, the probing model captures the effect of glitches by letting the attacker gather information on inputs like Y_1, Y_2 and also intermediate values like $X_2 Y_1, X_2 Y_2, X_2 Y_2$ etcetera. So, therefore, the attacker is now able to kind of probe inside the wires and basically is able to see the internal wires.

So, now the idea is that if you are still secured then; that means, that if you are combining the power leakages because of these intermediate results, you are still not able to or you should not be still able to do a do an attack. And therefore, right this is a very strong for attack model and therefore, if we can show the vulnerability or the protection against this kind of attack models, then essentially our designs are very sound.

So, now the question is like why is this design kind of vulnerable against you know like in the presence of glitches are X_2 , and note that here I assume that the value of Z_2 XOR of $X_1 Y_2$ essentially is shown over here and this is essentially I can take values like 0 and 1. So, basically I am kind of considering the inputs here, we are considering the inputs at this point which is essentially nothing, but Z_2 XOR of $X_1 Y_2$ ok. And the other parts essentially are shown here as $X_2 Y_1$, $X_2 Y_1$, and $X_2 Y_2$.

So, what we assume is so, if you want to read this table the way to read is that you see the value here, ok. So, suppose the value is 1, so that means, right we are basically assuming that there is a transition that Y_1 makes from 0, ok. So, all these are circuits, are all these values are pre charged to 0 and therefore, I if I write 1 over here that means, I am considering this glitch.

For example, I am considering this glitch which is essentially makes these transition like these from 0 to 1, ok. Likewise if I write Y_2 is equal to 0; that means, there is no transition because 0 remains a 0. If you consider X_2 then; that means, again there is a transition from 0 to 1 which is shown over here; if I write Z_2 XOR $X_1 Y_2$ is equal to 1; that means, again there is a transition here as shown over here. So, this point essentially makes the transition.

So, now, there is an interesting difference between the transition of X_2 and the transition of the remaining things because in this circuit we assume that X_2 comes little bit later say delta time difference later compared to the other glitches. So, that means, right if I consider the glitch Y_1 and X_2 then if Y_1 is coming here then the X_2 glitch comes a little later. So, it comes something like delta time difference later.

So, therefore, right here if you consider that the output is an AND gate, so, therefore, output is an AND gate. So, here a 0 gets processed with so, 0 0, the output is 0. So, therefore, the output becomes 1 only after delta time difference. So, therefore, this 1 becomes 1 after delta time difference. Whereas, this glitch is over here we have the point 0 for example, and that would imply that since this glitch comes later. So, therefore, the output of this XOR actually makes a transition like this. Because that we know that the two glitches right or the two input glitches are delta time difference apart and therefore there is a small time instance when both of these inputs become 1 and therefore, the

circuit becomes 1 over here and the output becomes 1 over here and then again comes back to the steady state 0.

So, likewise right if you consider the glitch here then this glitch is shown here. Now, here, there is no transition, and there is no transition because one of the inputs is held to 0, ok. So, therefore, this transition and there if I combine this with no transition then here also I get a transition like this. So, therefore, these transitions are at the instance delta likewise this instance is at that instance delta.

So, now if we consider the number of transitions before delta and at the time of delta for both the AND gates and the XOR are what are being shown here in this particular table. So, you can observe. So, let us see the AND first. So, the AND right essentially makes a transition at the instance of delta. This is the only transition which the AND gate is making. Here this AND gate is not making any transition, this AND gate is not in our account, but what we are considering is this AND and this AND. Out of this AND gate never makes a transition, but this AND makes one transition at the time instance delta.

So, therefore, we write here as 0 plus 1, which means like previously there was no transition and the time instance of delta there is one transition. If you observe the XORs then you can see that before also there was one transition from 0 to 1, here also there was one transition from 0 to 1. So, there were two transitions, that is 2 and after or at the time instance of delta there is again a 1 to 0 and a 1 to 0 transition. So, the again there are two transitions. So, this is 2 plus 2. So, likewise you populate this entire table.

And, now observe the average power for the AND gate, before the transition and after the transition. So, you will see here the average right here is 0 at the beginning and again 0 here, but after the transition; that means, at the instance of delta, here the sum is 2 plus 2 which is 4, which is essentially in the case when y is equal to 0, which is the actual unmask data. Likewise if you process it for y equal to 1, then you will see that again this essentially is 0 whereas, this sum is 1 plus 1 plus 1 plus 1 which is again 4. So, that means, right if I add up these ones then I get 4 and; that means, that the average is dependent on 4 transitions and 4 transitions. So, therefore, the glitch power is not leaking any information.

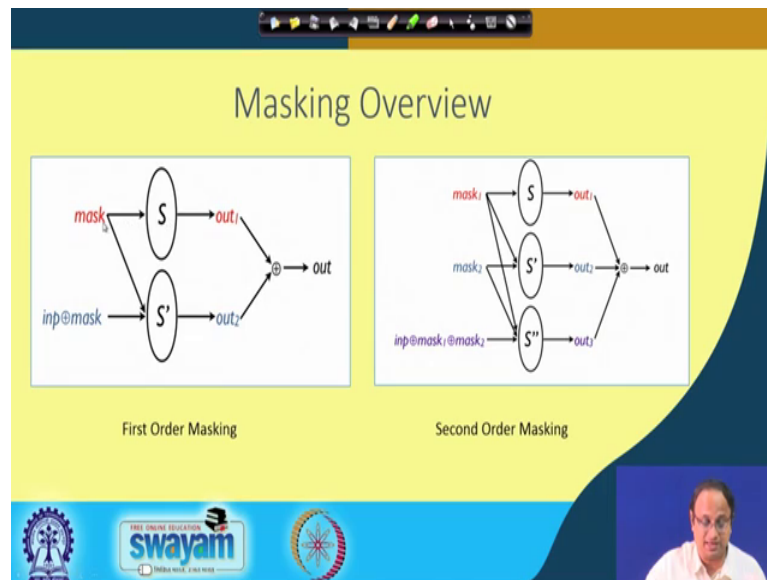
On the other hand if you do the exercise for an XOR, you see that before the glitch there is no problem because again you have got 2 plus 2 plus 2 plus 2 which is same as 2 plus 2 plus 2 plus 2 here in the case of y equal to 1, but for the case after the glitch right or after the transition you will see that the sum here is 1 plus 1 which is 2 and here it is actually 1 plus 2 plus 1 plus 2 which is essentially 6.

So, that means, right you see that now there is a dependence or here it was 2 and here it is 6, right. So, the average power is varying, and the average glitch power is varying and is also dependent upon your intermediate on you know like your unmask data. And, that implies that there is an average power consumption because of the XOR and the XORs are the reason or the culprit why this particular implementation or this particular masking is still vulnerable in the presence of glitches.

Because note that when you are doing the attack, you need not, you know like get into the circuit and try to count glitches actually, ok. The idea is that the glitch power being correlated with the with the with the unmask data, if we just do a power attack right, I will meet with much more observations you still will be able to get the key, and therefore, right I mean the whole point is that or the reason why this attack is per is working is because of the fact that the glitch or the glitch power is correlated to the unmask data.

So, therefore, I do would like to implement this in a way so, for these kind of attacks do not work, and that is essentially what we will see subsequently on how to perform.

(Refer Slide Time: 15:08)



So, likewise, right so, therefore, the overview on masking is as follows. So, we have got where we have got an input mask we in a First Order Masking we have got 2 input shares; that means, the input. So, we have got the mask the idea is that, there are 2 shares like S and S' are two parts which are processing.

So, there is one part which processes the mask as I said, independently which process on the mask and gives you these output out_1 and then the other part of the circuit which is S' , which actually process on the mask data where the input is XORed with the mask and it process on that part of the data and it gives you the out_2 , gives you the output. So, note that if you combine out_1 and out_2 you should be able to get out .

So, the idea here is that you observe also observed that S' process on the mask as well as on the input XORed with the mask, and it gives you the result which is out_2 , ok; whereas, at the part with is S right processes only on the mask wire. Likewise if you want to perform a Second Order Masking as you have seen that this masker or masking is vulnerable as a secondary analysis. If you want to protect it right then we would basically now break up the circuit into 3 parts.

So, S , S' and S'' where S and S' both processes on the $mask_1$ and $mask_2$. So, this process on $mask_1$ with the first part of the mask, this process on the second part of the mask; whereas, S'' processes on the input which is $mask_1 @ mask_2$. The input now is basically masked with the two masks; $mask_1$ and $mask_2$. So,

therefore, I get the corresponding masked result and therefore, this part processor not only this part of the input, but also which is the masked data, but also process processes on the mask values like mask 1 and mass 2 and gives you the third shared. So, now, the idea is that if I combine these three output shares I should get the corresponding result which is out.

So, of course, it has to be done in a much more careful manner because as we have seen that although theoretically the secured against first order attacks, but this is not secured in the presence of glitches.

(Refer Slide Time: 17:12)

The slide, titled "Traditional Masking: A Closer Look", features a logic diagram on the left and a mathematical expression in a box on the right. The logic diagram shows four AND gates at the top with inputs $\bar{x}y$, $b\bar{x}$, $a\bar{y}$, and abc . The outputs of the first three gates are connected to a single OR gate. The output of this OR gate and the output of the fourth AND gate are connected to a final AND gate, which produces the output z . The mathematical expression in the box is $z = \bar{x}y \oplus (b\bar{x} \oplus (a\bar{y} \oplus (ab \oplus c)))$. Below the diagram is the text "The Trichina AND Gate". The slide also includes logos for "THE OPEN UNIVERSITY" and "swayam" at the bottom, and a small video inset of a speaker in the bottom right corner.

So, again let us take a liquid look into you know like or take a closer look into this by considering just the traditional you know like masking of the of the AND gate which is the Trichina AND Gate. So, again you can observe that here there is a leakage. So, this is your traditional masking as we have already seen.

(Refer Slide Time: 17:28)

Vulnerability to Glitches

- Consider a glitch in input \hat{x}
- Table shows the number of gates affected by the glitch depending on the value of y
- The power consumption caused by the glitch is related to the number of gates affected
- The power consumptions differ with different values of y leading to the leakage of value of y

b	\hat{y}	y	AND	XOR
0	0	0	0	0
0	1	1	1	1
1	0	1	1	0
1	1	0	0	2

$\hat{y} = y \oplus b$

But, in the presence of glitches this is still vulnerable against attacks, as we have again seen, but let us try to redo it on the classical Trichinas gate.

So, here you observe that we consider inputs like so, basically we consider you know like the fact that y is essentially the XOR of \hat{y} which is y , you know like masked y with the mask. So, the mask here is b or denoted as b . So, again you know like, so, therefore, y is nothing, but the XOR or rather \hat{y} is essentially the XOR of y and b . So, y has been masked with b to give me \hat{y} .

So, now, if I observe this table; that means, that that would imply that if y is 0, then the masks are 0 0, the y is 1 then it is 0 1 or 1 0; likewise if the y is 0 it can also be masked as 1 and 1. So, now, if you observe right what happens right in this circuit when we consider that there is a glitch in the input \hat{x} ; so, \hat{x} is the mask data corresponding to x . So, this is very simple to observe that if I consider this glitch here and suppose I fix the input here.

So, let us consider you know like the last row in this table. So, the last row in this table tells me that you know like \hat{y} is held at 1 and it tells me that b is also held at 1. So, you see that I have held by hand as 1 and b as 1 and am considering the glitch \hat{x} . So, I am considering that there is a glitch here \hat{x} , ok. So, you can observe that if this is 1, then that would imply that this glitch will get propagated here. So, I will have a glitch here in this way. Likewise if there is a 1 here and a glitch here then this glitch will be

propagated here as shown here and likewise right here if I do not consider any activity in this part of the circuit then this glitch will also get propagated to the XOR.

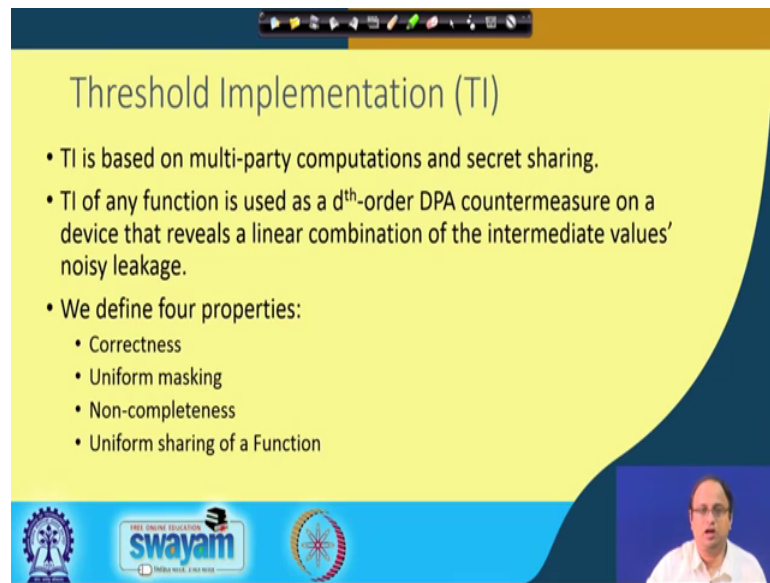
And, this glitch will and this glitch and this glitch we both will get propagated here as shown here in this glitch. So, now, if you consider therefore, you know as the number of glitches in the AND circuit; so, we will see that the and circuit there are two glitches, this one glitch and there is another glitch which is shown here. Again, XOR right there are two glitches which are shown over here. So, therefore, write here 2 and 2.

So, likewise you can try the other 3 cases and you can see that this is the kind of annotation of the number of glitches for the AND and the XOR. Again, right if you try to sit try to relate the glitches with the unmask data. So, in the unmask data here is the y . So, if we consider the cases for example, you know like y equal to 0 like y equal to 0 is in this case and y equal to 0 is in this case the AND gate right has got 0, 0 toggles here and two toggles here . So, if I take the average it is $0 + 2$ by 2.

If you consider here right this $1 + 1$, again the sum is 2, ok, right, but if you do the same exercise for the XOR you will see here it is $0 + 2$, but where it is, here it is $2 + 1$ which is 3, again it is different. So, again you will see that the number of glitches for the XORs are essentially getting correlated with your unmask data which is y . So, this again tells us that you know even the Trichinas AND gate is also vulnerable to glitches.

And, therefore, right is not even secured against first order differential power attacks. Of course, right I will require more observations, but still right the attack will work and this essentially is important to understand because you know like as we will see subsequently right we will be using this fact and we will be giving this fact in mind right we will basically try to construct a better design which is essentially called as TI or Threshold Implementations.

(Refer Slide Time: 21:07)



The slide 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 title 'Threshold Implementation (TI)' is centered at the top in a large, dark font. Below the title, there are four bullet points. The first two points describe the basis of TI and its use as a DPA countermeasure. The third point introduces four properties, which are listed in a sub-bulleted format. At the bottom of the slide, there are logos for 'swayam' and other educational institutions, along with a small video inset of a man speaking.

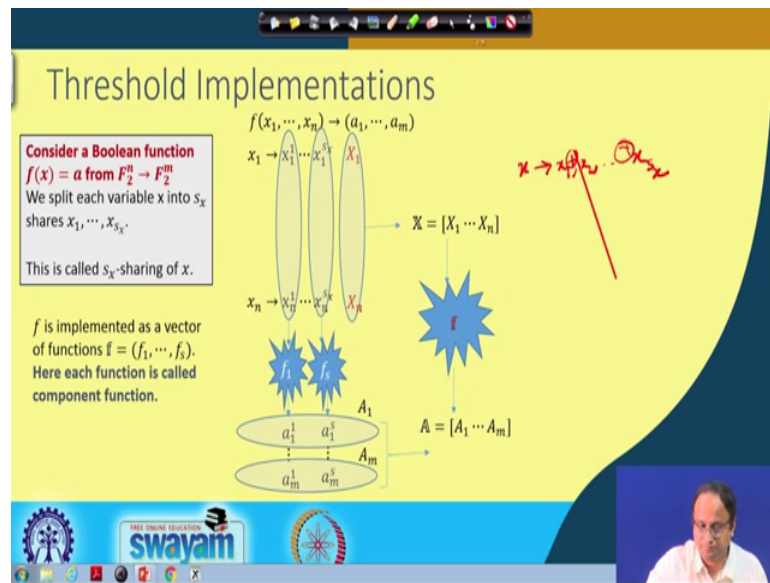
Threshold Implementation (TI)

- TI is based on multi-party computations and secret sharing.
- TI of any function is used as a d^{th} -order DPA countermeasure on a device that reveals a linear combination of the intermediate values' noisy leakage.
- We define four properties:
 - Correctness
 - Uniform masking
 - Non-completeness
 - Uniform sharing of a Function

So, the idea of threshold implementations is based on multi-party computations and secret sharing. The idea is that TI on any function is used and the basic objective is that if I take any function and then suitably apply a TI or Threshold Implementation then it can be used as a d^{th} order DPA countermeasure on the device that reveals a linear combination of the intermediate values noisy leakage.

Of course, right there are four important properties that a TI needs to satisfy and I will be you know like one by one defining those criteria. They are called as Correctness, Uniform masking, Non-completeness and Uniform sharing of a Function. So, we will try to see how these kind of properties are defined and how we develop on them one by one.

(Refer Slide Time: 22:03)



So, to start with right what is the basic idea of a threshold implementation? So, suppose I have got a function which is $f(x)$ which basically kind of takes n -bit value and transforms it into another m -bit data. So, the idea is that now we will split each variable x into s_x shares, ok. So, like we were having d share. So, suppose you know like I use S_x to indicate the number of shares that I use for my input data x .

So, basically; that means, that if I take x right I basically split x into split x into x_1, x_2 like this right to x_{s_x} . So, there are s_x shares of x , ok. So, that means, if I XOR these things right I should get back, I should get the original data. So, so, therefore, right I mean with this you know like sharing. So, therefore, if I take an n -bit input like x_1 to x_n then basically means that I break up x into X_1 . So, these are you know like my individual shares, ok. So, x_1 has been broken up into X_1^1 so on to $x_1^{s_x}$. Likewise right x_n has been broken up into X_n^1 to $x_n^{s_x}$, and this you can indicate as now as a vector say X_1 to X_n .

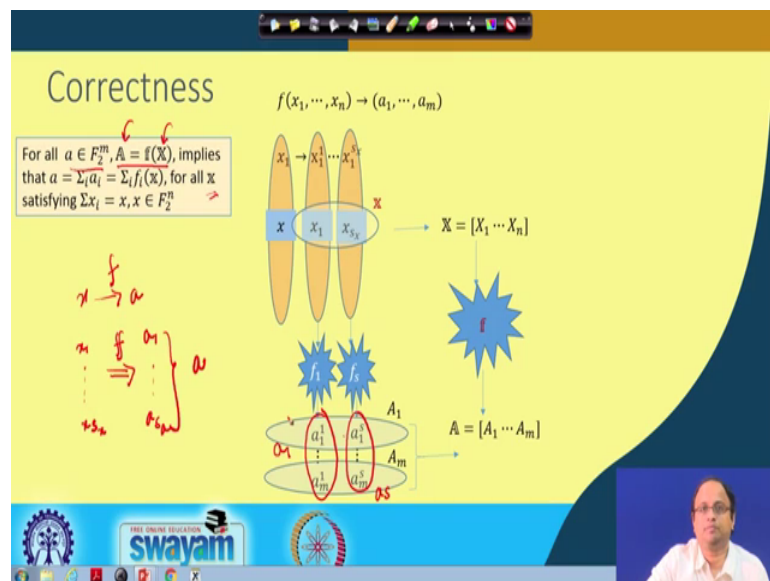
So, this vector X_1 to X_n is basically therefore, every element in this vector X_1 is therefore, having s_x shares. So, now, what we want is that we would like that the in the function f right you will also be broken up into say s shares f_1 to f_s , such that every functions say f_1 will now operate on these shares and will give me a vector a_1 , ok. So, therefore, for example, f_1 right will give me a_1 . So, remember that it will also

process of n-bits and basically right give me a result which is a 1 1 to a m 1. Likewise the f s circuit or the f s part of the circuit will also give me an output which is a 1 s to a m s.

And, now what you can observe is that you can now treat these things like a 1 to a s right as s shares and I can indicate that as say the vector A 1, ok. So, therefore, right I will have A 1 to A m such shares and that gives me my vector A as shown over here. So, this is my vector output vector A. So, therefore, what we do is basically we take this math cal X and math cal A to indicate the vectors for the input and the vectors for the output and the function f is also a vectorial function because now it is a combination of s functions.

So, therefore, right we call this so, therefore, the idea is that if s is implemented as a vector of functions as shown here as. So, the vector f which is a combination of f 1 to f s and the whole idea is that each of these functions is therefore, called as a component function.

(Refer Slide Time: 25:14)



So, you can also you know like simplify these notations slightly by taking that and now I will represent this x 1. So, this is x 1 is my input for example, this was this was my input right here as we have seen here I had my inputs x 1 to x n. I can represent this entire input as the by the variable X, ok. So, I can represent this entire variable, this entire part by the variable x and each of these shares like you know like X 1 1 to X n 1; so, this is my first component that we had I can represent that as x 1, ok. So, likewise I will have x s x because there are s x shares.

So, therefore, now, what I will do consider; so, therefore, this essentially is my sharing of the input x . So, remember that each of these x 's here are not 1-bit values, but n -bit values now. So, therefore, right if I can kind of generalize this and say that. So, likewise right the output also will be masked in a similar way. So, therefore, now I will have f_1 to f_s . Each of these data will now as we have seen is giving was giving me you know like a 1_1 to a 1_s , and a m_1 to a m_s .

So, you can also imagine that these right essentially like for these components, like suppose you are you are you are observing this components say a 1_1 to a m_1 , you can observe these as you know like a 1 for example. So, what I mean is that you can you know like consider these part and write that as say a 1 and likewise right there are s shares. So, you can write that as say you know like a s ok. So, this is your s th share.

So, therefore, right what it means is that for all a 's like for a is your m -bit value, and suppose what you can you can write this input. So, basically now this function f takes the input shares indicated as X or $\text{math cal } X$ and this function gives you an output share, ok. So, this output share is indicated as A or $\text{math cal } A$ and this implies that you know like if that a equal to $\sum a_i$; that means, if I add up these a_1 to a s , ok; that means, I add up the output of each of these functions ok, for all these input shares, then this should satisfy you know like I mean then right I should get the output a .

That means, right, if x gets mapped into a right by the function f then; that means, right for all the input shares that I have for x , right which is entirely denoted as say x_1 to s x for example, they if I apply now the vectorial function; the vectorial function is suppose integrated as f math cal then I will get output shares like a 1 to say s a for example, so, such that if I combine them I get the original data a , ok. So, that is essentially the basic idea behind thresholding. Of course, right we have to see how to implement it, so that we essentially get security against the power attack adversary. So, we basically define the properties that we need in order to obtain that, but we will take that up in the next class.

Thank you.