

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

Lecture – 30
Power Analysis – VI

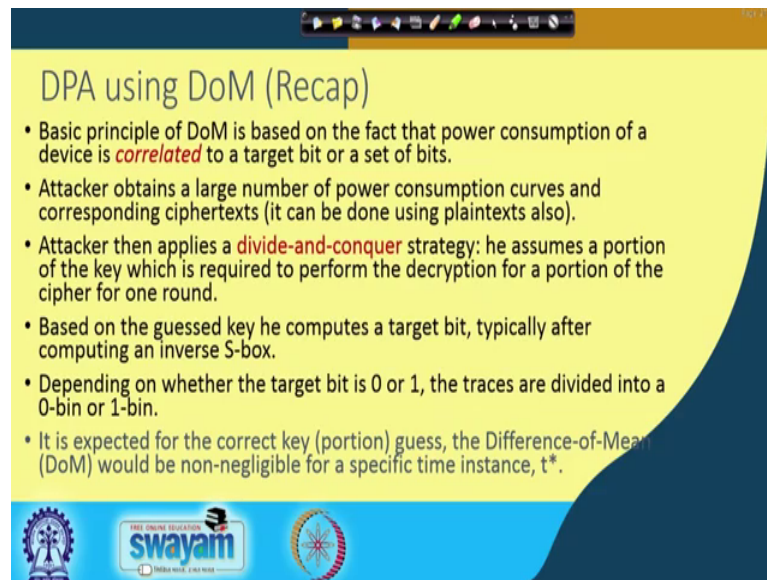
So, welcome to this class on Hardware security. So, we shall continue our discussions on power attacks and Power Analysis.

(Refer Slide Time: 00:23)



So, we were discussing in the last class on difference of mean attacks. So, we shall be discussing about how to implement different difference of mean technique on simulated tracing traces. We shall be discussing about a new technique next about which is called as Correlation Power Attacks or CPA and we shall also be discussing about how to implement CPA's on CPA on simulated traces.

(Refer Slide Time: 00:45)



The slide is titled "DPA using DoM (Recap)" and contains a list of six bullet points. At the bottom of the slide, there are three logos: the Swamyam logo, the logo of the Ministry of Education, Government of India, and the logo of the National Institute of Education, India.

- Basic principle of DoM is based on the fact that power consumption of a device is *correlated* to a target bit or a set of bits.
- Attacker obtains a large number of power consumption curves and corresponding ciphertexts (it can be done using plaintexts also).
- Attacker then applies a *divide-and-conquer* strategy; he assumes a portion of the key which is required to perform the decryption for a portion of the cipher for one round.
- Based on the guessed key he computes a target bit, typically after computing an inverse S-box.
- Depending on whether the target bit is 0 or 1, the traces are divided into a 0-bin or 1-bin.
- It is expected for the correct key (portion) guess, the Difference-of-Mean (DoM) would be non-negligible for a specific time instance, t^* .

So, this is a brief recapitulation of what we already discussed. So, essentially how difference of mean works is essentially the basic principle of it basically based on the principle that in DoM right, we try to kind of exploit the fact that power consumption of a device is correlated to a target bit or set of bits and the attacker essentially obtains a large number of power consumption curves and corresponding cipher text, it can also be done from the plain texts.

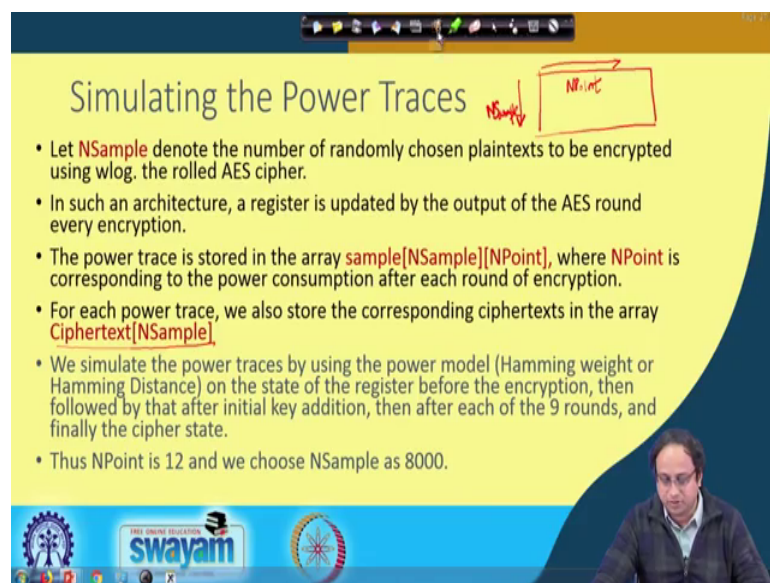
But suppose it is done from the cipher text which is probably more I would say intuitive and then that, attacker basically applies a divide and conquer strategy. So, he assumes a portion of the key which is required to perform the decryption for a portion of the cipher of 1 round say the last round for example,. So, he basically guesses a part of he basically sees the cipher text cases are a part of the key and does say 1 round of decryption and then, based upon the guessed key, he computes a target bit. So, typically this is after computing an inverse S-box.

Like suppose, I want to calculate the input of the last round S-box, I need to perform an inverse S-box on the exclusive or of the portion of the cipher text and your and the portion of the key which you are guessing and because you do not know what is the key. So, you guess that byte of the key for example, and then you calculate the inverse S-box and then, you basically find out any or you target some specific bits. So, this could be for

example, the LSB that you target; it could also be the MSB or it could be any linear combination of the bits also.

So, depending upon whether the target bit is 0 or 1 and that depends upon your guess of the key, the traces are divided into a 0 bin and a 1 bin ok. And it is expected that for the correct key or correct key means the portion of the correct key, the difference of mean would be non negligible for a specific time instance which we say t^* that is t^* is the time when actually the computation is being taking place is essentially happening.

(Refer Slide Time: 02:49)



Simulating the Power Traces

- Let $NSample$ denote the number of randomly chosen plaintexts to be encrypted using wlog. the rolled AES cipher.
- In such an architecture, a register is updated by the output of the AES round every encryption.
- The power trace is stored in the array $sample[NSample][NPoint]$, where $NPoint$ is corresponding to the power consumption after each round of encryption.
- For each power trace, we also store the corresponding ciphertexts in the array $Ciphertext[NSample]$.
- We simulate the power traces by using the power model (Hamming weight or Hamming Distance) on the state of the register before the encryption, then followed by that after initial key addition, then after each of the 9 rounds, and finally the cipher state.
- Thus $NPoint$ is 12 and we choose $NSample$ as 8000.

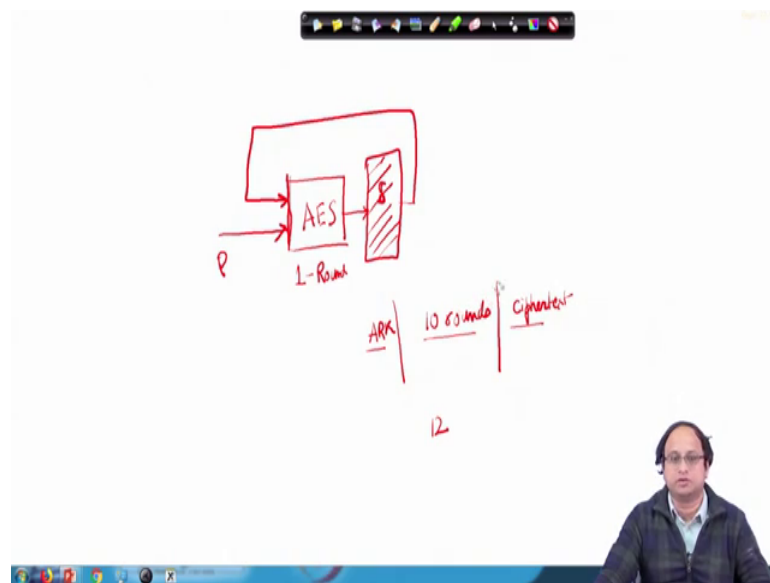
So, therefore, right I mean 1 because you know like sometimes when you are starting to work on or DPA for example, it is a good it is an it probably is a good idea to kind of start with something which are called as simulated power traces. So, you basically try to simulate the power trace and develop your technique. Later on right you can of course, replace the simulated power traces by the real power traces in which case the attack will become more difficult because you will be having noise and other phenomenon , but to develop your technique and to understand your technique, right simulated power traces is a good starting point.

So, let us start with that you know like with such kind of power traces we will be simulating. So, suppose you are let us consider a rolled AES architecture; a rolled AES architecture as we have seen means that there is 1 round of AES and you are basically kind of iterating it to get all the 10 rounds of AES. So, it is basically an iterated

architecture and you basically what you are trying to do is you are basically trying to kind of calculate the power consumption of these device.

So, now, of course, like when you are actually doing this attack you will be having the actual set up and hence, you will be getting it , but what we will be trying to do is in this or considering in this case is we will be trying to get a simulation or simulated power trace. So, let us see how we can do that.

(Refer Slide Time: 04:19)



So, what I mean to say is essentially this like now you have this rolled architecture which means you have your AES block; this is the AES round and the AES round the output of the AES round suppose you are storing in essentially a register this register S essentially storing the corresponding result of your encryption. So, basically like this is not the complete AES; so, this is just say 1 round of AES and it is being iterated.

So, I am not drawing a very accurate diagram, but this is kind of indicative that is initially you have you basically taking the plaintext and of course, like there is a key that you need to kind of XOR and it you are you start your operation of AES; AES and AES round and then every clock cycle that you do like essentially you know that there are 10 rounds of AES operations. So, basically you can you are essentially latching some data onto this register ok.

And because of the dynamic switching's which are happening in this register you are expending power, the architecture is consuming power right. So, therefore, you can imagine that this state register I mean although this very hypothetical figure, essentially right are it is basically consuming power in 12 distinct time instances ok. The first time it is expending power is when you are doing the initial add round key that is you are doing the initial add round key operation at that point it is expending a power and then, there are 10 rounds of AES operation ok. And finally, you are getting the cipher text.

So, typically right that is essentially your 1 plus 10 plus 1. So, there are 12 time instances or 12 instances in time when the register essentially is making a toggle and therefore, there is a power which is being consumed. So, what we basically try to model in our simulated environment is we basically see the register content and suppose you have a very simple model like hamming weight or hamming distance based upon that you kind of predict the power consumption ok. Note that when you are doing this prediction you have a 128 bit register. So, you are basically considering the 128 bit register and based upon that right you know that the hamming weight.

For example, could be from 0 to 128. So, there could be like 129 possible power levels and therefore, right depending upon the content of that register content of the 128 bit state registered you kind of indicate a power consumption. You say that my AES for this particular time instance is consuming such amount of power ok. So, this is again a simulation and we will we essentially try to use this simulated framework to develop our technique or experiment on our technique.

So, therefore, right I mean with this kind of you know like simulated power trace, we would like to basically now get into how the d DoM works. So, therefore, right imagine that I do NSample number of such operations ok. So, NSample typically means that you essentially have you know like you are basically encrypting plaintext for example, suppose you are encrypting say you know like 1000 plaintext. So, therefore, NSample would denote the number of randomly chosen plaintext which you are encrypting ok. So, without.

So, basically like you are encrypting say n number of inputs and that n is essentially here NSample ok. Now in such an architecture in such a rolled architecture as I said a register is updated by the output of the AES round after every encryption and the power trace is

again. So, you now you are basically storing the power trace in a register I mean you are basically storing it in array and the array is name is say sample NSample NPoint.

So that means, basically right is essentially it is a 2 dimension as you can see it is a 2 dimensional array. So, you can visualize this as you know that you are basically storing the power in the form of a 2 D array, we are essentially suppose you are using this index to you know like say the number of the sample like suppose you are encrypting the first or the second or the third and so on and this axis essentially is storing the point or the tiny instance of your AES operation.

So, this is your this direction is your NPoint and NPoint as we have seen for our simulated environment is say 12 because there are 12 time instances when the register is making a toggle. There is an initial add round key followed by the 10 rounds of operation and finally, when you are getting the cipher text. So, for each power trace, we also store the corresponding cipher text because we also need the cipher text and the cipher text essentially as you can understand is that if there are NSample plaintext that, you are encrypting you get NSample cipher texts ok. And that is essentially denoted by this array which is cipher text NSample.

So, we simulate that we simulate the power traces by using the power model. So, the power model could be like hamming weight or hamming distance on the state of the register before the encryption and then, followed by that of the initial key addition; then after that there are like nine rounds of encryption and finally, there is a cipher state that you are generating.

So, therefore, the NPoint is 12 and we choose NSample something like say 8000 ok. So, this is an ad hoc choice, but suppose in you know like your NSample is 8000. So, now, we clear this ok.

(Refer Slide Time: 10:03)

A Simple Trick

- For where the power model is Hamming Weight, a simple trick can be applied.
- Let the attack target one of the key bytes, **key**.
 - Thus a corresponding S-Box is targeted.
- We denote the corresponding ciphertext byte in the array `Ciphertext[Nsample]` as **Cipher**.
- For each of the `Nsample` runs, the analysis partitions the traces, `sample[Nsample][NPoint]`, into a 0-bin and 1-bin, depending on the target bit.
- For estimating the target bit, the attacker computes the Inverse S-Box on the XOR of the byte **Cipher** and the guessed key byte, **key**.
- One may observe that the ciphertexts for which the ciphertext byte corresponding to a target byte is say **Cipher**, will map into the same bin.
- Thus the traces can be stored in a smaller array `sample[NCipher][NPoint]`, where `NCipher` is the number of cipher bytes, which is 256 in our case.

Logos for IIT Bombay, Swamyam, and IIT Madras are visible at the bottom of the slide.

So, now so there is a simple trick that you can observe that particularly this works when you have your hamming your power model is hamming weight power model. So, when the power model is hamming weight a single trip can be developed. So, the let the attack target 1 of the key bytes ok. So, let me call that as key for example. Now that is it therefore, you basically target the corresponding S-box. So, this S-box could be say the first S-box of your last round.

So, we denote the corresponding cipher text byte in the array cipher text `Nsample` as cipher ok. So, again this is the first byte of your cipher text. For each of the `Nsample` runs that you are doing, the analysis partitions the traces into 0 bin and 1 bin and it targets the you know it basically depends upon the depending on the target bit it partitions into 0 and 1 bin. Now, for estimating the target bit the attacker computes the inverse S-box on the XOR of the byte cipher and the guessed key byte.

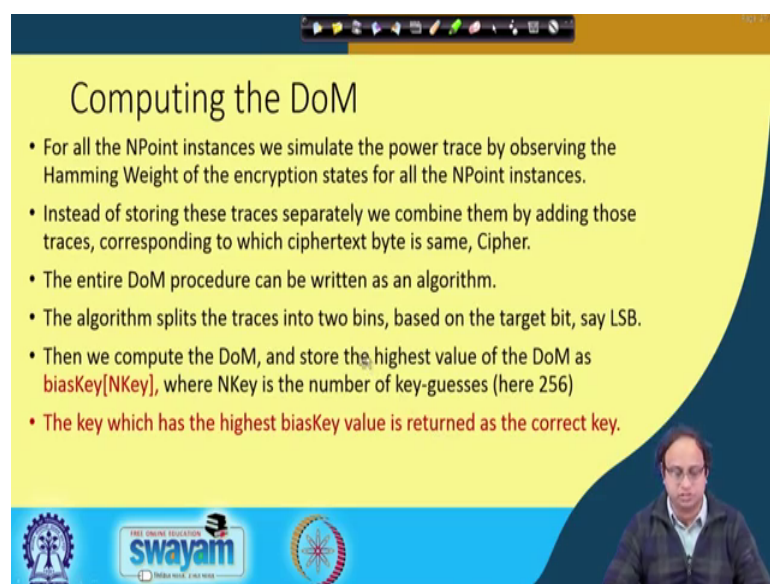
Because we know that cipher is your corresponding cipher text byte and the corresponding key which you are guessing is a key you XOR them, calculate the inverse S-box find out your target and you split and you put your trace into either the 0 bin or into the 1 bin. But note a point that the cipher text for which the cipher text by it corresponding to a target byte is a cipher will map into always the same bin. So, what I mean to say is that even if the for example, if you consider AES there are 16 bytes in the cipher imagine the zeroth byte for example,. So, if the zeroth byte is a constant

irrespective of what is the value of the remaining 15 bytes that power trace will go to either the 0 bin or will go into either the 1 bin.

So, that implies that you can basically you know like rather than storing the power trace into the array sample n cipher NPoint which is I mean rather than storing it into the array that we saw previously which was sample NSample NPoint because you had like 8000 encryptions. You can now store the same information in a in a smaller area you can store it in say sample n cipher NPoint and n cipher in our case is something like 256 because that byte can take only 256 values ok. So, you need not store that in an array you know like sometimes when you are attacking for example, as we will see right counter measures, you may need millions of traces you may need millions of encryption.

So, you need not have such a big array, but you can actually store that into a much smaller array depending upon the size of your word ok. So, you can store it in for example, sample say 256 NPoint because what you are basically doing is you are kind of partitioning your entire power trace it into 256 partitions and all the members of that partition the whether it will go into the 0 bin or whether it will go into the 1 bin irrespective of key guess actually will go into the same mean ok. Because their fates are same so therefore, right you need not store that in such a bigger array. So, this is a simple trick that you can try to adopt when you are developing your implementations.

(Refer Slide Time: 13:11)



Computing the DoM

- For all the NPoint instances we simulate the power trace by observing the Hamming Weight of the encryption states for all the NPoint instances.
- Instead of storing these traces separately we combine them by adding those traces, corresponding to which ciphertext byte is same, Cipher.
- The entire DoM procedure can be written as an algorithm.
- The algorithm splits the traces into two bins, based on the target bit, say LSB.
- Then we compute the DoM, and store the highest value of the DoM as **biasKey[NKey]**, where NKey is the number of key-guesses (here 256)
- The key which has the highest biasKey value is returned as the correct key.

Logos at the bottom: IIT Bombay, swayam (Free Online Education), and another IIT logo.

So, how do you compute the DoM? So, for all the NPoint instances we simulate the power traces by observing the hamming weight of the encryption states for all the NPoint instances that means NPoint is again 12. So, for all these 12 instances you basically you know like you can simulate the power traces. So, I am talking about the simulation right now. So, you can simulate by observing the hamming weight of the encryption states for all the NPoint instances and instead of storing these traces separately you can combine them by adding those traces corresponding to which the cipher text byte is say cipher ok. So, what you can do is because later on you need to calculate the average.

So, what you can do is that for all those k all those traces or for all those corresponding encryptions where the zeroth bytes say takes a takes a specific value, you essentially can just add on to its current content; current content in the array where you are storing the power traces and you can store the sum. So, you can store the sum and therefore, the you can essentially you know like optimize your storage and the entire DoM process. Therefore, you know like as we will see you can actually write that in a very systematic manner in a nice form of an algorithm which you can code and which you can automate.

So, the algorithm basically splits the traces into 2 bins based on the target bit say LSB and then, we compute the DoM and store the highest value of the DoM as an array which is say bias key. So, the number of entries in bias key will be how much? Will be again the number of possible key guesses and you have got 256 key guesses for say the zeroth key byte.

For every possible guess you guess a bias that bias is nothing but the difference of mean and as we have discussed previously that if you have a correct key guess, then there should be an NPoint; some time instance when you should get a non negligible value for the difference of mean ok. And if you get a negligible value of difference of mean in all instances, then it is quite likely that that is not your candidate key. So, therefore, the key which has got the highest bias key, we value should be returned as a possible candidate key.

(Refer Slide Time: 15:23)

The slide is titled "Algorithm for Performing DPA". It contains the following elements:

- Code:** A pseudocode algorithm for DPA. It starts with input parameters: sample, NSample, NPoint, Cipher, and Ciphertext. The output is binKey. The algorithm iterates over cipher bytes (0 to N-1). For each cipher byte, it calculates a partial cipher by applying an inverse S-box. It then iterates over all possible key bytes (0 to 255). For each key byte, it calculates the XOR of the partial cipher and the key byte, and then calculates the Hamming weight of the result. The Hamming weight is compared to the Hamming weight of the ciphertext byte. The difference is used to calculate a bias. The bias is used to calculate the bin index. The bin index is used to calculate the bin key.
- Graphs:** Two graphs showing "DPA bias values" vs "trace# (*1000) ->". The left graph shows a sharp peak at trace# 0, indicating a strong bias. The right graph shows a much flatter distribution, indicating a weak bias.
- Text:** Two text boxes explaining the simulation. The first box says: "DPA on first key byte of 10th Round of AES. Power is Simulated by Hamming Weight of the Registers after each Round." The second box says: "Power is Simulated by Hamming Weight of the Registers after each Round, with superimposed Gaussian Noise."

So, therefore, this is essentially the whole idea and that essentially has been written in the form of an algorithm and the algorithm essentially does nothing but essentially does or you know like implements whatever we have discussed till now ok. So, basically if you read and this you can read in details from the textbook that we sight and you will find that what it basically does it calculates an XOR of the cipher and the key. So, the cipher is set a zeroth byte and key is say 1 byte of the of the key, you kind of XOR them. You calculate the inverse as S-box and then you basically go to the input of the S-box and then, that is what you called as partial cipher ok. Then, what you do is you extract the LSB of partial cipher.

So, you do an anding with say 1 you can extract any other bit also for that matter, but suppose we extract the LSB because we based on the LSB we will now partition the traces and if that bit is say 1; that means, if that bit is 1; then what we do is that we start incrementing the counters for the 1 bin ok. So, we basically have a sum bin that means, the summation of all the power traces for bin 1, we increment that. We also increment a counter; that means, how many traces are going into that particular bin and note that your sample array is now indexed by cipher because as I said that you essentially can index it by the 256 values ok.

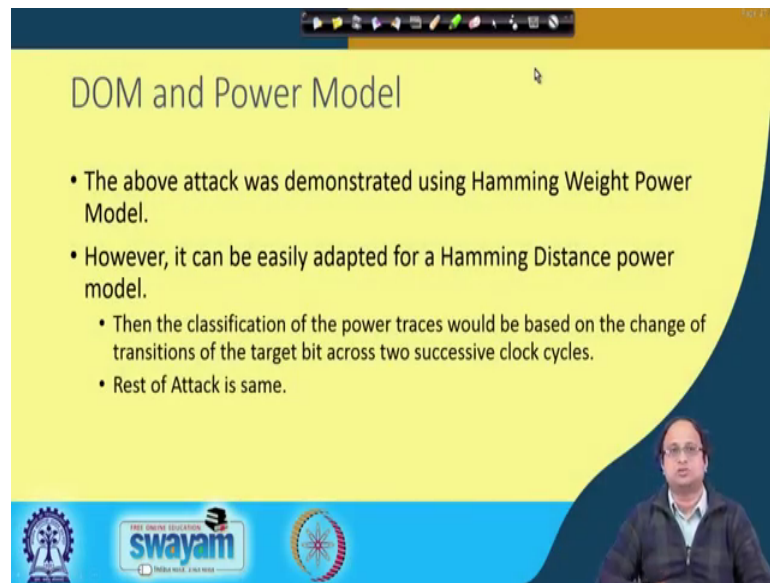
And then, if that bit is 0; then you do similarly for the 0 bin. So, basically you calculate the average and finally, you calculate the difference of the averages and that is stored in

this mean diff array and then, you find out the absolute mean I mean you find out the you know like the maximum absolute value and the idea is that whichever gives you the maximum value is basically returned as the corresponding correct key. So, here is an experimental run on this and you can very well simulate it because if you have an implementation of AES even without going to a set up actually which may be costly and not always reachable. You can actually easily simulate the power traces and you can try this algorithm. You can just try to maybe implement it using a simple c.

So, is exactly that is being done here. It is basically done on the simulated environment where there is no noise; that means, the power essentially is modeled by the hamming weight of the 128 bit state register and then, we apply these difference of mean technique and then, we target the last round that means, the 10th round of AES and we try to do a DPA on say the first key byte ok. As you can see that this shows how the bias value kind of increases for the current key whereas, it does not for the wrong keys and therefore, you can easily distinguish the correct key byte from the wrong key bytes. And therefore, it shows that the difference of mean attack works in this setting. More realistic simulation would be where you can add a gaussian noise to your hamming your to your power model to your simulation.

So, you can you know use any library which generates these kind of noises and you can kind of embed it or superimpose it on your hamming weight values and now if you try to do a difference of mean, as you can see that the attack becomes more difficult, but nevertheless you can see the separation ok. So, the separation is not as wide as here, but we still get a separation and we are able to retrieve the correct key.

(Refer Slide Time: 18:35)



DOM and Power Model

- The above attack was demonstrated using Hamming Weight Power Model.
- However, it can be easily adapted for a Hamming Distance power model.
 - Then the classification of the power traces would be based on the change of transitions of the target bit across two successive clock cycles.
 - Rest of Attack is same.

swayam

So, therefore, that brings us to the you know like the link between difference of mean and power model. As we have discussed priorly also that the power models can be of different types for example, like hamming weight, you have got has the hamming distance power model. So, the question is right how can you adopt your technique for hamming distance power model and if you think right then it can easily be done so. The only change that you have to do is that now the classification of the power traces would be based on the change of the transitions rather than the absolute hamming weight ok.

So, you have to basically calculate the hamming weight , but the hamming weight will be on the XOR of the present state with the previous state. So, therefore, if you again target the last round of AES, then you can see that the state register initially holds the output of the ninth round and after the final round it gets replaced by the tenth round output which is the cipher. So, therefore, if you take an XOR between the cipher and the ninth round input right I mean the ninth round output sorry. So, if you take an XOR between the cipher and the ninth round output and find out the hamming weight of that, then essentially you can again do the same thing ok.

So, the rest of the description remains exactly the same. So, you can easily adapt your technique and make it work when you are you know like modeling your power using say hamming distance power model.

(Refer Slide Time: 19:57)

The slide is titled "Stochastic Power Models" and contains the following text:

- The DoM technique for DPA is non-profiled.
- On the contrary, there could be another kind of side channel attacks which fall into the category of profiled attacks.
- In profiled attacks, one needs to have access to a pair of identical devices:
 - Target: limited control and running cipher with fixed key.
 - Profiler: full knowledge and control of the input and keys.
- Profiling can help to stochastically learn the power model more accurately.

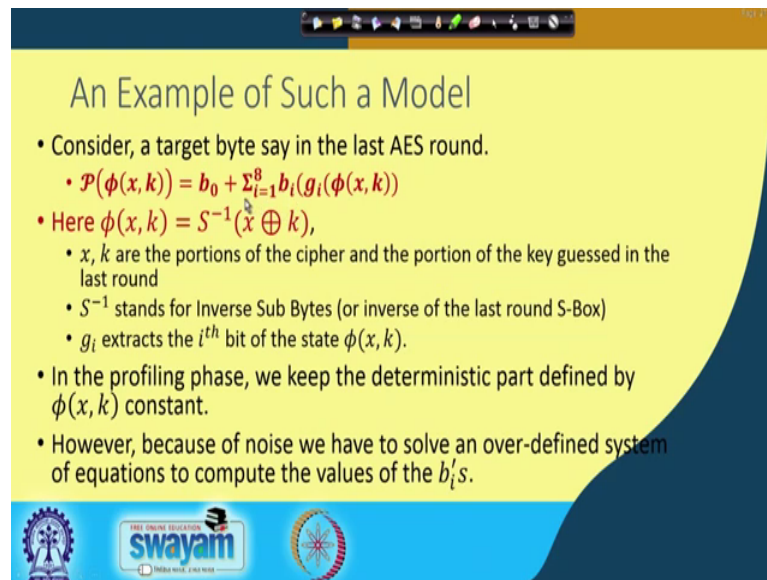
The slide also features a video inset of a man speaking in the bottom right corner, and logos for Swamyam and other institutions at the bottom.

But here is a very key important point; that means, hamming weight and hamming distance although we have been using; one of the reasons we are using them is because they are easy to use. So, there is something which is called as stochastic power model which basically is also or may be mentioned here. So, the difference of mean technique as we have seen for DPA is actually what is called as a non profile attack ok. So, non profile means that the attacker essentially gets the hardware, where the secret key is embedded and starts to do the attack right away ok.

As opposed to this there is another category of side channel attacks which are called as Profile attacks. In profile attacks one needs to have access to a pair of identical devices. So, there are 2 identical devices; the first one is a target which is essentially where you have got limited control and there is a running cipher say AES with a fixed key. You cannot change that key ok. On the other hand, in a profiling stage or a profiling phase you have been provided with a similar hardware exactly identical hardware where you have full knowledge and you have got control on all the inputs like you can change the inputs, you can change the key and you can do profiling as much as we want.

So, profiling can be used to stochastically learn the powered model more accurately and that will probably help you to improve the accuracy of your power attack.

(Refer Slide Time: 21:17)



An Example of Such a Model

- Consider, a target byte say in the last AES round.
 - $\mathcal{P}(\phi(x, k)) = b_0 + \sum_{i=1}^8 b_i (g_i(\phi(x, k)))$
- Here $\phi(x, k) = S^{-1}(\hat{x} \oplus k)$,
 - x, k are the portions of the cipher and the portion of the key guessed in the last round
 - S^{-1} stands for Inverse Sub Bytes (or inverse of the last round S-Box)
 - g_i extracts the i^{th} bit of the state $\phi(x, k)$.
- In the profiling phase, we keep the deterministic part defined by $\phi(x, k)$ constant.
- However, because of noise we have to solve an over-defined system of equations to compute the values of the b_i 's.

So, for example, like if you consider a target byte say the last round of AES. So, the. So, let us write by this equation what we are trying to do is so we are trying to say consider x which is a portion of the cipher and k or small k as a portion of the key which you are guessing in the last round and then, suppose you basically what you are doing is you are calculating the inverse S-box which is denoted here as S inverse ok. So, you are calculating S inverse of x XOR with k and this I denote as say ϕ of x comma k .

So, this ϕ of x comma k is my deterministic portion of my power consumption because it kind of can be determined. It is not a noisy component. It is also called as signal as we will see later on ok. So, it is a signal component of your power consumption. So, when you calculate this ϕ of x comma k , what you do is you apply a function subsequent to that which is say g_i or denoted as g_i which does nothing but extract an i th bit as we have seen in our algorithm right we had extracted the LSB.

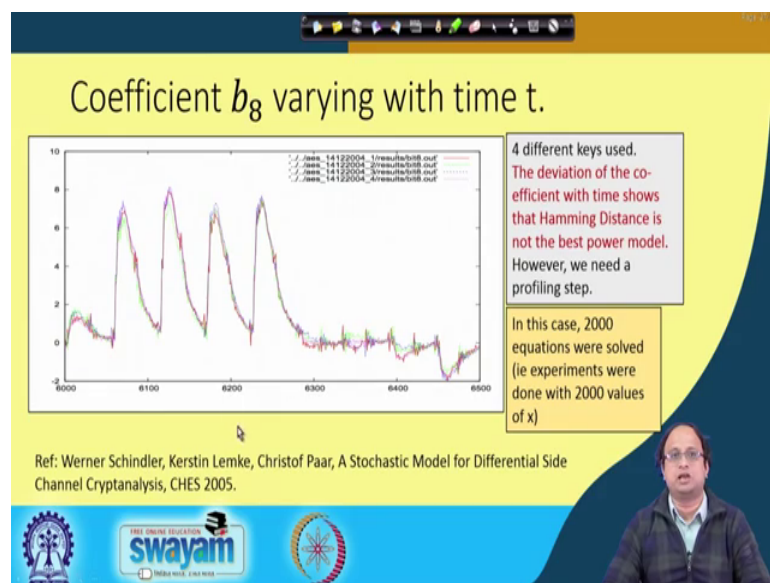
So, g_i basically tries to extract any one of the bits say the i th bit of the state ϕ x comma k and now you essentially have a say a register or a byte for example, and you want to predict what is the corresponding power consumption because of that content. So, what we are doing previously is that we were assuming a hamming weight for example; right and hamming weight would mean that every bit is equally contributing ok.

But on the other hand, as we know that in an electronic circuit right lot of weird things can happen; what can basically happen is that each of the bits are basically contributing

in a different manner and that essentially is captured by this equation, where we write that the power consumption is modeled as $\sum_{i=1}^8$ because there are 8 bits positions and each of these contribute or each of the contribution of the bits are each the way the each of the bits are contributing are modeled by this coefficients say b_i ok.

So, you have got b_1, b_2, b_3 and so on till b_8 and you also have b_0 which basically stores any other component. So, in the profiling phase, we keep the deterministic part defined and we keep it kind of constant by keeping $\phi(x)$; we keep the x and k constant and we essentially try to you know like observe different power values. Note that because of noise there will be you know like still certain differences and therefore, write you as you probably will have you know like an over defined system of equations. You probably need an over defined system of equations to solve the values of these b_i 's.

(Refer Slide Time: 23:55)



So, here is an experimental you know like description written by this paper by Schindler and other co-authors, where they try to develop a stochastic model for differential side channel crypt analysis. So, you can observe that here there is an experiment which has been done with 4 different keys and the deviation or the of. So, here we plot say the you know like say. So, here is a plot of the eighth coefficient that is b_8 for example, and we show how b_8 varies over time.

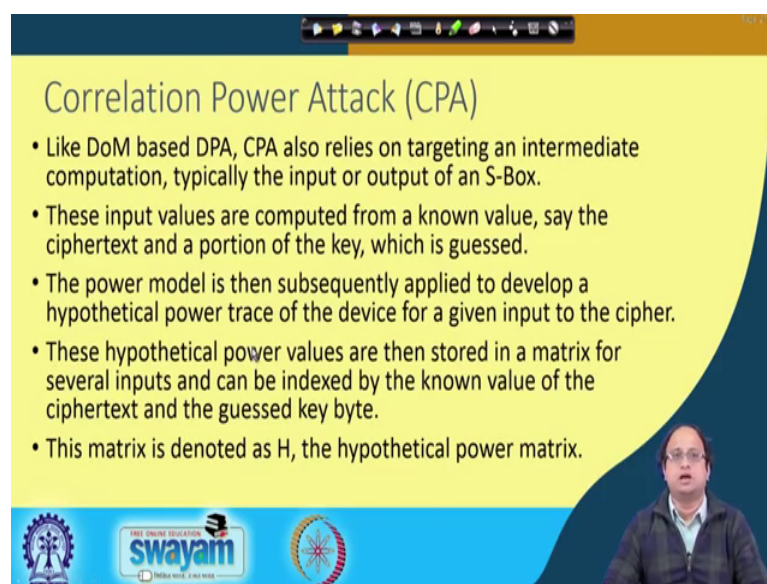
So, you can easily see by that you know the by this variation of b_8 that if you are applying a hamming weight power model, then that is not accurate because in a in a in a

hamming weight power model we are assuming that the contribution of any of the bits right is kind of constant overall run over all time instances. But you can observe here that that is not so and also another interesting point is that even if you are changing the key, the deviation is actually minor; that means, there is a small deviation in you know in your prediction of the corresponding contribution.

So, that implies that you know like you need not always you know like know the correspondingly correct key, you can very very well work with any guess ok; any guess of the any guess of the key byte and that would give you quite a good estimate about what is the contribution of the bits and therefore, you can you know like understand what are what is the stochastic model of your device and later on right you can for example, in this case another point to be noted is that here there were 2000 equations which we have solved and therefore, right you need an over defined system of equations.

But after doing this effort right you have got a better understanding about your leakage model which you can try to exploit in your next phase of the power attack and that is why right these power attacks are very strong ok. But at the same time right you need an access or an opportunity to do this profiling phase. So, wherever it is possible we should do this ok.

(Refer Slide Time: 25:55)



Correlation Power Attack (CPA)

- Like DoM based DPA, CPA also relies on targeting an intermediate computation, typically the input or output of an S-Box.
- These input values are computed from a known value, say the ciphertext and a portion of the key, which is guessed.
- The power model is then subsequently applied to develop a hypothetical power trace of the device for a given input to the cipher.
- These hypothetical power values are then stored in a matrix for several inputs and can be indexed by the known value of the ciphertext and the guessed key byte.
- This matrix is denoted as H , the hypothetical power matrix.

swamyam

So, now we will start discussing about correlation power attacks or CPA. So,. So, I will just you know like initiate this discussion and we will we will see more details

subsequently. So, in a correlation power attack this is exactly or you know like quite similar to the difference of mean based attack. Because here also you are relying on the fact that there is a you are basically targeting an intermediate computation and typically the input or output of an S-box just like the DoM technique.

These input values are computed from a known value again which is could be like the set of cipher text and a portion of the key which is guessed and the power model again you know like so again you are using a power model say again, we are using the hamming weight power model and then, you are using it subsequently and in applying their applying it to develop now a hypothetical power trace.

So, in the previous case we were kind of trying to correlate the power trace with the single bit or to a linear combination of bits ok. But in this attack we will be trying to kind of develop a hypothetical power trace and try to correlate it with a real power trace and the and we were to see how we can develop the data structures and how do we define our correlations to better understand how the attack works.

So, this we will take up in the next session and we will continue our discussion.

Thank you.