

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

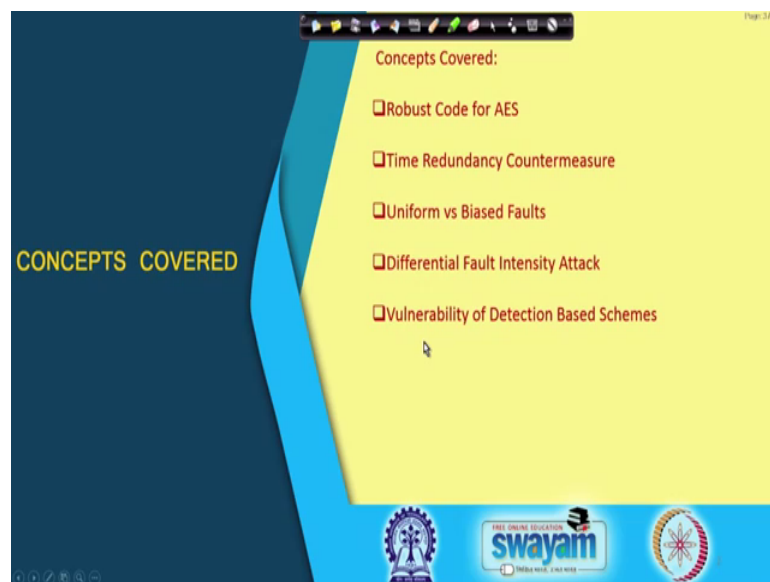
Lecture - 51

Redundancy Based Fault Countermeasures and Differential Fault Intensity Attacks

(Contd.)

So, welcome to this class on Hardware Security. So, we shall be continuing our discussion on fault tolerance with respect to cryptographic implementations.

(Refer Slide Time: 00:23)



So, we shall be starting to discuss with where from we left last class that is Robust Code for AES, we shall be talking about Time Redundancy Countermeasures and also Biased Fault attacks and eventually on differential fault intensity attacks.

(Refer Slide Time: 01:45)

Linear Codes:
 V : binary linear (n, k) code with $2k \geq n$.
 H : $\left[\begin{array}{c|c} P_{(n-k) \times k} & I_{(n-k) \times (n-k)} \end{array} \right]$ over $GF(2)$.
 Check matrix $(n-k) \times n$.
 G : $\left[\begin{array}{c|c} I_{k \times k} \\ P_{(n-k) \times k} \end{array} \right]_{n \times k}$.
 $HG = [P | I] \cdot \left[\begin{array}{c} I \\ P \end{array} \right] = P + P = 0$.
 m (message) \xrightarrow{G} codeword Gm .
 Syndrome = $HGm = 0$.
 If because of an error, $c = c + e$.
 $H(c) = H(c + e) = H(c)$.
Non-linear Codes The linear code V is converted to a non-linear code: (n, k) . C_V .
 Number of undetectable errors = 2^k .
 Number of detectable errors with prob 1: $2^n - 2^k$.

So, we know that you know like when we talk about linear codes, we basically very briefly right has got. So, we have got a binary linear n comma k code for example,. So, this is an binary linear n comma k code with say $2k$ which is greater than equal to n .

So, these are you know like useful constructs which are used in the context of linear codes and we have got you know like a what is called as a check matrix and this is a very usual matrix which is used and this has got two components. One is say P of n minus k cross k and that is followed by an identity matrix which has got a dimension of I n minus k cross n minus k and this is usually defined over $GF(2)$; that means, the elements here are binary elements.

So, this you can easily understand has we will have got the dimension of n minus k cross n . So, now, if you take this matrix the idea is that, you essentially can define a corresponding matrix which is called as the generator matrix which is essentially denoted as in this way there is suppose this is I k cross k and that is followed by say P n minus k cross k ok. So, this totally will have dimension of n cross k . So, you can easily verify that if I do H into G like if I multiply H into G then I will get this as kind of you can write this as P concatenated with I multiplied with I and followed by P .

So, this is basically P plus P . So, therefore, this is equal to 0 . And so, what we do is that suppose there is a message m and we basically kind of transform this message into a corresponding code word for the message by you know like applying your generator

matrix G and basically getting Gm for example,. So, now, you can basically see that, what we do is that basically kind of compute what is called as a syndrome on the code word by taking these Gm and applying the H matrix on it that is applying the check matrix on it. Now since H of G is equal to 0 I get the syndrome as 0 .

On the other hand if right there is an error if because of an error if because of an error what happens is that, this code word gets kind of polluted to something like c plus some error e then when you apply the check matrix on c tilde you get H of you know like you basically have c plus e and as we know that H of c will be equal to 0 ; we get what is called as H of e and this is essentially a non 0 term and that kind of kind of alarm such that there is an error.

So, now the question is right and without going into details because this is you know like subject in itself; we know that you know like the number of undetectable faults or you know like undetectable errors is equal to 2 power of k and the number of detectable errors there is detectable errors actually you know like interestingly with probability 1 because for sure they can be detected, we have got something like the remaining thing that is 2 to the power n minus 2 to the power of k that is all of them would be detected, but at the same time there are 2 to the power of k errors which are not detectable. So, then here we you know like bringing the concept of what is called as non-linear code. So, this is an example of this was an example of linear codes. So, in non-linear codes you basically try to kind of you know like increase the or rather decrease the number of undetectable errors I will be right.

You basically kind of you know like for the remaining errors which you can detect you can afford some amount of probability of missing them. So, the idea is that here you basically take the; so the linear code for example, the linear code V is converted to a converted to a non-linear code to non-linear code, again I denote it as n comma k and write that as C V So, I will give an example which we often use in the context of cryptography and.

(Refer Slide Time: 06:44)

$$C_V = \{(x, w) \mid x \in GF(2^k), w = [P x]^3 \in GF(2^r)\}$$

Clearly, error (e_x, e_w) .

$$(x+e_x, w+e_w) \in C_V$$

$$[P(x+e_x)]^3 = [P x]^3 + e_w$$

$$e_w \text{ will be missed by this undetected}$$

$$2^k \begin{cases} \text{No. of undetectable faults: } 2^{k-r} \\ \text{No. of errors detectable with prob. of 1: } 2^{k-r} \\ \text{No. of errors detectable with prob. of } 1-2^{-r}: 2^{k-r} \end{cases}$$

So, this example essentially is or in the context of fault tolerance for cryptography is based. So, let me let me denote it as C V; where C V is nothing, but x comma w where x suppose belongs to GF 2 to the power of k. So, that is for example, in AES there is 2 power of 8 and then I calculate this w by applying a matrix transformation denoted as P on x and then performing finite field cube.

So, this essentially belongs to GF 2 to the power of r. So, there is kind of a complexion and then followed by in like converting this into GF 2 to the power of r. So, clearly you can see that which kind of errors will basically get missed. So, if you observe that. So, that would mean that if there is an error. So, consider there is an error and this error is denoted as say e x and e w.

So, that would mean that suppose I have got x and w here, I basically make an error in x and I make an error in e w such that this resulted term also fault faults in this code word then it cannot be detected. So, for example, what will happen here is that this will become x plus e x, and this will become w plus e w and; that means, that now the code word for x plus e x will be P applied on x plus e x and then I do an whole cube on that and that should be equal to w right plus e w; that means, you know like w plus e w if that happens right then it cannot rejected because this is the corresponding code word for it, but then this also becomes a valid code word and therefore, this also belongs to C V. And what is w? W is nothing, but P x whole cube plus e w.

So, if you basically make an analysis of this equation and see like how many are possible solutions and which essentially can you know if any solution for this will be missed. So, therefore, in error which essentially kind of satisfies this equation will be missed by this code word. So, therefore, right if I just summarize the result, here we will see that the number of undetectable faults right. So, remember right in the previous case right it was; in this case right this is 2 to the power of k minus r and the number of errors which are detectable with probability 1 probability of 1 is equal to 2 to the power of n minus 1 minus or plus 2 to the power of n minus 1 minus 2 to the power of k minus r and the number of. So, interestingly right you will have some errors or number of errors which can be detected detectable with a probability of 1 minus 2 to the power of minus r plus 1 .

So, that is not exactly one, but still right there will be some cases which will like;. so that there is like 2 to the power of n minus 1 minus 2 to the power of k minus 1 ok. But the number of undetectable faults is 2 to the power of k minus r and if you remember right it was previously 2 to the power of k in that case of linear codes and that essentially gets significantly reduced.

And that is why right people have tried to apply such kind of techniques and get interesting designs. So, now, with this background right we can basically see the design. So, here you can see that this is your cubic function which has been used for example, there is a cubic function here used and here is a cubicle function has been used, but before that there is a compression and L-Compress.

So, what are this? So, the L-Compress or the compress as I have said just now; so if you remember like the in the mix columns what did we did basically? We basically found out the parity of every byte there and the compress function is nothing, but it basically kind of XORs the parity of each of these columns.

So, you saw that you know like it does not change for the mix columns and therefore, I do basically get a mix column for I mean a parity for this column that is denoted as L_0 , now this is denoted as L_1 and this is denoted as L_2 and this is denoted as L_3 . So, totally we have got you know like you know like you have got a parity for 4 bits here, you have got a parity I mean for 1 byte here and so, on right and therefore, if you basically taken XORed of all these things right and then you get L_0 . So, these total set is essentially

what do you get at the output of this complex function and that is followed by an L-Compress and this L-Compress is nothing by a map.

So, therefore, imagine that this is say you know like say r bits for example; you would like to compress this into $r L$ where $r L$ is less than r . So, therefore, you just kind of multiply this output by a matrix whose dimension is say r cross $r L$. So, and then you get this result which is $r L$ and then you perform the usual cubic operation and then you basically check. So, this essentially together stands for something like your P as I just now denote it and likewise here. So, then you finally, do a check and the idea and the results say that there is a significant improvement because of this, and this is essentially leveraged in some of the designs.

(Refer Slide Time: 12:38)

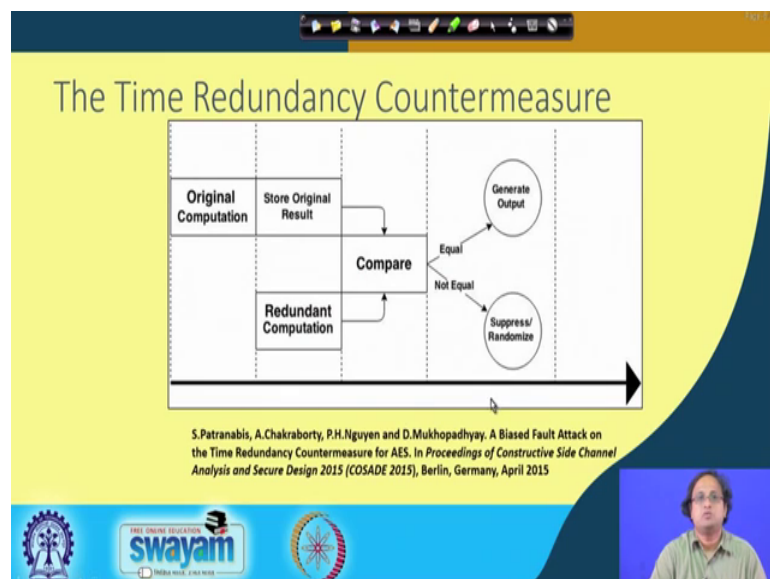
The slide is titled "Nonlinear Codes for Information Redundancy". It features a flowchart on the left side illustrating the process. The flowchart starts with an "In" input, which goes through a "RegX" block. The output of "RegX" is split: one path goes to an "Enc" block, and the other goes to an "L-Predict" block. The "Enc" block's output goes to a "Compress" block, which then goes to an "L-Compress" block. The "L-Predict" block's output goes to an "L-Compress" block, which then goes to a "Cubic" block. The outputs of the "L-Compress" blocks are compared in a decision diamond labeled "=?". If the comparison fails, it leads to an "Error" block. If it passes, it leads to an "Out" block. There are also two text boxes on the right side of the slide. The first text box states: "The cubic network increases the complexity of encoding and decoding in a quadratic fashion, but also decreases the number of undetectable faults significantly." The second text box provides a source: "Source : Guo, Mukhopadhyay, Jin, Karri, Security analysis of concurrent error detection against differential fault analysis – Journal of Cryptographic Engineering, 2014". A third text box at the bottom right mentions: "Mark Karpovsky et. al., Dependable Systems and Networks (DSN) 2004." The slide also includes logos for "swayam" and "Information Redundancy – Robust Codes" at the bottom.

(Refer Slide Time: 12:40)



So, therefore, the idea is that the; so the question is right there are several detection techniques as we have seen, but the broad the bigger question is right does detection always guarantee security.

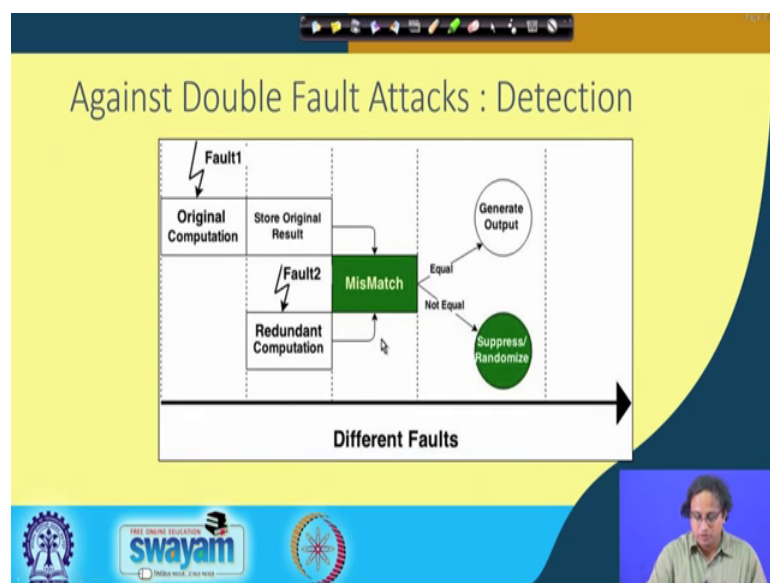
(Refer Slide Time: 12:49)



So, here we will bring the idea of what is called as we will discuss in the taking one of the redundancy techniques, which is very popular and that is the time redundancy countermeasure. So, as we have seen that here we do an original computation. Computation we again do it by re do it by a redundant computation and then we check.

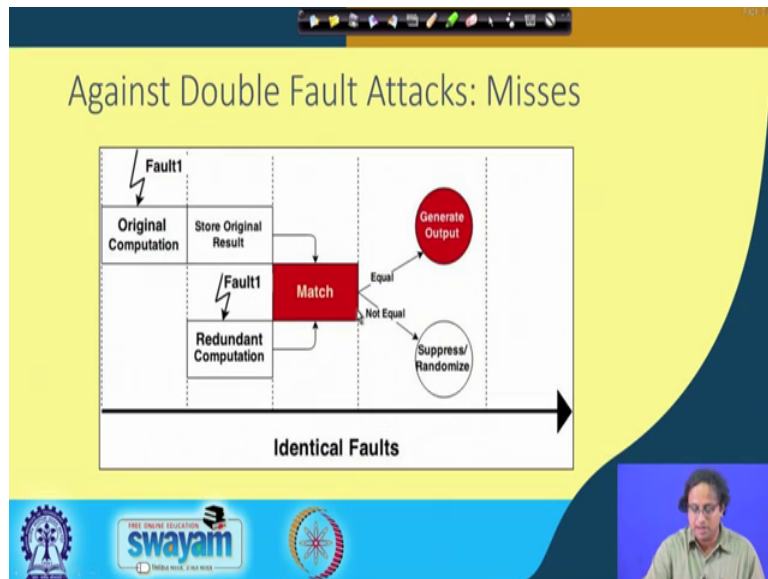
So, they can be two strategies I can take; if it is not equal. So, one of them could be that I suppress the output, but a more or better way would be to randomize the output. So, the attacker even does not know right because even if you suppress an output; that means, the attacker knows that there has been a fault in one say that is also an information leakage. So, a better strategy would be to randomize the output, but; however, we know for most part of our discussion right I will assume that the output has been suppressed. But it can be kind of extended to even for that case where the output has been randomized.

(Refer Slide Time: 13:41)



So, what we do is that, if there is a mismatch then we basically you know like suppress or randomize the output, but if there is a match then we generate the output.

(Refer Slide Time: 13:48)



So, then the you know like the reason one of the reasons why we kind of show here as you know like red here is that, this is fine because you know like if you are suppressing output than attacker does not get any information, but what we need to analyze is this situation. Where there is a match right and that record gives an output. So, now, remember that this match is kind of assumed by the designer is that if there is a match the designer is kind of assuming (Refer Time: 14:17) there has been a no fault, but in reality what can actually happen is that there are two faults and these two faults are cancelling each other.

(Refer Slide Time: 14:27)

Beating The Countermeasure

- Improving **fault collision probability**
 - Enhancing the probability of identical faults in original and redundant rounds
- Two major aspects
 - The **size** of the fault space
 - The **probability distribution** of faults in the fault space
- A smaller fault space enhances the fault collision probability
- A non-uniform probability distribution of faults in the fault space also enhances the fault collision probability

Handwritten notes and formulas:

- $P_1^2 + P_2^2 + \dots$
- $\frac{f_1 f_2}{P_1 \dots P_N} = \frac{f_N}{P_N}$
- $\sum P_i^2$
- $Var = \frac{\sum P_i^2}{N} - \left(\frac{1}{N}\right)^2$
- $\sum P_i^2 = N \cdot Var + \frac{1}{N}$
- $P_{coll} \uparrow \propto N \cdot Var + \frac{1}{N}$

So, one can argue you know like what would be the probability of such kind of fault collision. So, imagine that you know like you know like you have got two possible stay you know like possible status states and you would like to calculate the probability of fault collision; that means, right you would like to kind of calculate the probability of two identical faults in the original as well as in the redundant round.

So, you can easily work that out, imagine that suppose you have got you know like possible faults denoted as say f_1 to f_N . So, this is your all possible faults and the probability of inducing each of the faults are suppose denoted as p_1 to p_N .

So, then we know that the probability that you basically inject the same fault in both occurrences can be found out by the sigma of π square. Because you know like it is basically like you are you are kind of you know like inducing the same fault suppose you are inducing the fault belong in f_1 in both the redundant as well as the correct operation or the actual operation; that means, in the original and the redundant round that probability would be p_1 square likewise similarly for f_2 this will be p_2 squared. So, therefore, if we add it up right we will get sigma into π square.

Now observe that are variants of the fault distribution also can be written as sigma of π squared by N , suppose there are N possible faults and this is equal to nothing, but minus 1 by N whole square so that means, that sigma π squared can be easily calculated as N multiplied by the variance plus 1 by N . So, therefore, the probability of the collision; so let me write this as probability of collision is N multiplied by the variance plus 1 by N .

So, now note that when the variance is 0; that means, when you have a uniform fault distribution, then this probability would be equal to 1 by N ; that means, like suppose you have got an if you consider the AES state matrix where there are you know like 2 to the power of 128 possible faults right then this probability is negligibly small. But at the same time one should be careful that if I observe a variance; that means, if I kind of increase a bias in my faults distribution, which can be measured by these variance statistic then I can actually increase the probability of collision significantly.

So, therefore, if the variance increases then the probability of collision also increases. And that essentially something which is kind of stated over here as the non-uniform probability distribution of faults in the fault space, with enhance the fault collision probability we would like to therefore, take care of such kind of scenario.

(Refer Slide Time: 17:14)

Uniform Fault Model

- All faults are equally likely

The diagram shows a vertical list of faults $f_1, f_2, f_3, \dots, f_N$. A bracket groups f_1, f_2, f_3 as F_1 (Fault for Original Computation). Another bracket groups $f_4, f_5, f_6, \dots, f_N$ as F_2 (Fault for Redundant Computation). A cloud contains the equation $\Pr(F_1 = F_2) = 1/N$.

So, therefore, right if you take this into account. So, therefore, right this is the classical scenario where all faults are equally likely then this probability because the variance is 0, the probability of fault collision is equal to 1 by N which can be small; however, right what we what we can observe is that if we.

(Refer Slide Time: 17:22)

Biased Fault Model

- A total of n faults possible under a fault model F
- Each fault f_i has a probability of occurrence $\Pr[f_i]$
- Let V be the variance of the fault probability distribution
- Degree of Bias of a fault model increases with increase in V

Fault Model	$\Pr[f_1]$	$\Pr[f_2]$	$\Pr[f_3]$	$\Pr[f_4]$	$\Pr[f_5]$	$\Pr[f_6]$	$\Pr[f_7]$	$\Pr[f_8]$	V
1	0.125	0.125	0.125	0.125	0.125	0.125	0.125	0.125	0
2	0.225	0.200	0.175	0.125	0.100	0.075	0.050	0.050	0.004
3	0.500	0.250	0.125	0.050	0.050	0.025	0	0	0.026

So, here is an example to show you know like that the variance is a measure of this biasness in the fault distribution. So, imagine that you have got a fault model where all the faults right like f_1 to f_8 occurs with equal probability then the variance is 0.

Whereas, you can observe that this is a more biased fault injection where as this even more because there is certain faults which never occurred then the variance increases further. So, therefore, variance is useful measurement for the degree or bias in the fault model, and as we observe right then the priority of fault F_1 equal to F_2 is equal to this directly proportional to the variance. So, therefore, if the variance increases this probability of collision also increases.

(Refer Slide Time: 18:06)



So, what is the adversarial of perspective of this? So, you have got a precise fault model versus a biased fault model and the question is right whether you can exploit this.

(Refer Slide Time: 18:19)

Fault Intensity

The impact of fault varies with the tuning of the parameters of the fault inducing process.
More true for low cost equipment.

Insertion of Fault through Clock Glitches:
With increase of clock frequency more bits start getting affected.
We say the fault intensity increases!

Nahid Farhady Ghalaty, Bilgiday Yuca, Mostafa M. I. Taha, Patrick Schaumont:
Differential Fault Intensity Analysis. FDTC 2014: 49-58

So, this brings us you know like what is called as fault intensity and gradually we will this define what is called differential fault intensity.

So, this is based on the work which is published an FDTC in 2014 which define what is called as fault intensity. The idea is that if you take a; say let us consider a combination of logic is got a 4 input and 4 bits output and then gradually we start to increase the clock glitch; that means, we try to kind of increase the you know like try to induce a fault by you know like a clock glitch for example. Similarly I can do it for maybe other mechanisms like voltage glitch and so on but let us consider a clock glitch here and you can see that, the fault which you are observed in the output right is not really random which we have been assuming in the in the context of differential fault attacks.

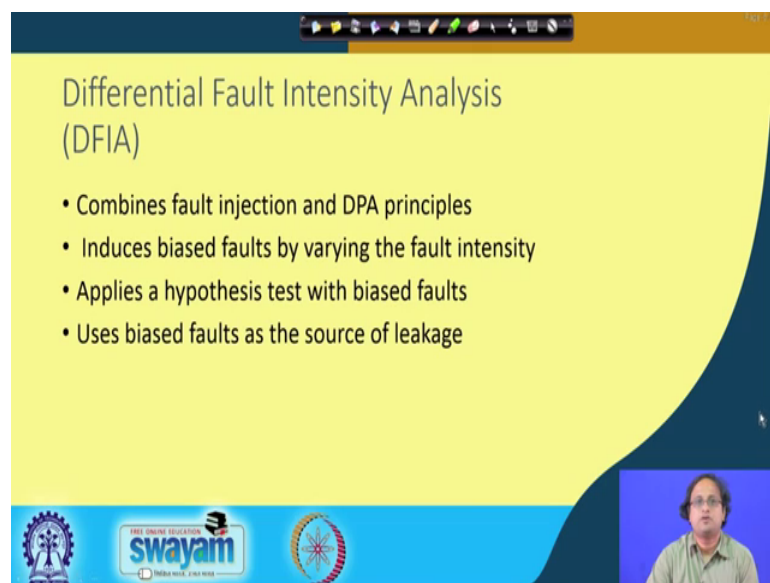
So, what is shown here is that if you are increasing the clock frequency for example, the first line which fails right is the critical which is in the critical path the means we just got the maximum latency. If we increase it further right it is most likely than the next one will fail and therefore, right if we even if you repeat this pattern, you would expect a probable you know like a probable repeatable pattern.

And more importantly using the clock injection frequency, you can actually control the number of bits which are getting affected. For example, if the after I have done a bit of characterization, I can you know like probably inject the clock frequency or operate it in a situation where I get you know like faults which are like this, which is like a single bit

faults for example, or maybe you know like faults which has got 2 bits maximum, which I give which get corrupted.

That means, right I am trying to kind of induce faults which are biased and not really random and that is a very important thing. Because you know like when we talk about classical fault tolerance, then we basically assume that all faults are equally probable because you know like in nature, largely the faults are kind of unbiased. But when we talk about fault attacks or fault tolerance in the context of cryptography, one thing which we have to keep in mind is that the adversary can accordingly manipulate the fault injection and hence can bias the fault injection. And therefore, we have to kind of revisit the fault tolerance techniques, keeping these kind of biased nature in the fault distributions.

(Refer Slide Time: 20:29)



Differential Fault Intensity Analysis (DFIA)

- Combines fault injection and DPA principles
- Induces biased faults by varying the fault intensity
- Applies a hypothesis test with biased faults
- Uses biased faults as the source of leakage

swamyam

So, therefore, right this brings us to what is called as differential fault intensity attack, it basically combines fault injection and the principles of differential power attacks or that we have as we have seen original side chain attacks. It basically induces bias faults by varying the fault intensity, it applies a hypothesis test with bias faults and uses biased faults as a source of leakage.

(Refer Slide Time: 20:52)

So, let us see how it works. So, there are two steps in this particular attack; in the first step right you basically do a fault injection which is biased. So, this would probably happen more right in case of when you have got a low cost fault injection techniques, like voltage glitch or clock glitch or even in the case of maybe an Nem injection; you will see that which is not very controlled well. So, you will see that for example, you will have what f_1 to f_Q all these are different fault intensities right; and when you are injecting the fault what is happening is that the state is getting corrupted and therefore, you are getting say Q such injections and all of these injections have gone you know like apparently a biased nature in the faults. That means, probably you are getting you know like by a fault here which is you know like maybe a 1 bit corruption or 2 bit corruption, but it is not random; that means, all faults or not occurring here with equal probability.

So, therefore, you get correspondingly faulty cipher texts; note that this attack right is very kind of significantly different from the classical DFA in this in this aspect. In classical DFA we basically do an attack where we have got the place we have got the faulty cipher text and we also have the correct cipher text, but here these attack works only on the faulty cipher text.

So, imagine that are got like C_1 to C_Q like and all of them are faulty and therefore, they are denoted as C_1 dash to C_Q dash so. So, now, there is a step 2 of this attack in the step 2 of this of this attack we do a hypothesis testing with biased faults. So, what we do

here is that. So, this extraction is something like that have a side channel attack or side channel analysis. So, you basically guess the correct key and you and it helps in observing the bias in the fault distribution. So, the idea is that if you are able to correctly guess this key then when you kind of invert back; that means, we XOR the cipher text and you take the inverse SBOX and you come to the location or the site of the fault injection remember here was the site of the fault injection and the input of this SBOX.

You should be able to see these biased. And if you are basically making a wrong guess then you will not see this bias you will basically see a random distribution. So, a useful; so, you will basically for that you need a distinguisher as we used in the context of DPA. So, one of the very common forms or distinguishes which is used here is this, that is you basically guess the key that is shown here as K tilde, you take the faulty cipher text, we applied the s box inverse and you find out this S tilde and then you accumulate all of them and then you basically take the sigma of the hamming distances of this.

The idea would be that this hamming distance right you are basically assuming that you are operating in a region where this hamming distance is small. So, therefore, you would assume that this sigma that you know like that should be minimized when you are correctly guessing the key; that means, for the correct guess of the key this particular parameter sigma of the hamming distance of S tilde should be minimized.

So, therefore, you can again you know like applying a similar to what we have done in the DOM or in the correlation attacks; you can basically kind of arrange the keys or rank the keys with respect to which basically minimizes this parameter. And the one with minimize is most you know which gives the minimal output is the candidate key.

(Refer Slide Time: 24:00)

Attack on the Time redundancy Countermeasure

- All faults are restricted to a **single byte**
- Two kinds of fault models
 - Situation-1: Attacker has control over target byte**
 - Situation-2: Attacker has no control over target byte**
- Control over target byte makes fault model **more precise** but is **costly to achieve**

Suitable

Symbol	Fault Model
FF	Fault Free
SBU	Single Bit Upset
SBDBU	Single Byte Double Bit Upset
SBTBU	Single Byte Triple Bit Upset
SBQBU	Single Byte Quadruple Bit Upset
OSB	Other Single Byte Faults
MB	Multiple Byte Faults

Fault Model	Faults Possible(n)	
	(Situation-1)	(Situation-2)
SBU	8	128
SBDBU	28	448
SBTBU	56	896
SBQBU	70	1120
OSB	93	1488

Logos: swayam, MEDIA WISE, 2019 WISE

So, likewise right you can; we will see how we basically experimentally verify it. So, we basically kind of considered two situations. In situation-1 we will assume that the attacker has got no control on the target byte and in the situation-2 we will assume that the attacker has got or in the first situation the attacker has got control over the target byte. In that situation-2 the attacker has got no control over the target byte. So, now, what we observe here is that, we basically considered you know like different kinds of fault models. So, for example, the fault model can be a single bit upset, it can be a single byte double bit upset, it can be a single byte triple bit upset it can be a single byte quadruple bit upset and there can be other kind of byte faults also possible. In particular let us consider about the SBU or which is the Single Bit Upset fault model.

So, of course, you can understand that if I have got a fault model which is more precise than it is also more costly to achieve. So now, we basically again as I said that we basically kind of characterize our device and we see that, in situation 1 where we assume that the attacker has no control as got control over the target byte I need lesser fault injections to get my required faults. Whereas, in the situation 2 where I am basically more haphazard in my fault induction I require more faults, but at the same time right this is pretty manageable and within this faults right I am able to create such kind of fault injections.

(Refer Slide Time: 25:34)

The Fault Injection Set-Up

- Time redundant AES-128 implemented in Spartan 3A FPGA
- Fault injection using clock glitches at various frequencies
- Xilinx DCM to drive fast clock frequency
- Internal state monitoring using ChipScope Pro 12.3

So, now what we basically considered these and see how it kind of helps us to do the attack. So, this is the hardware that we will be considering again remember this is the just a recapitulation of our old design is an iterative architecture, we again use an on you know like an arbitrary function generated to inject the first clock and then we basically use the ChipScope Pro to basically monitor the internal faults.

(Refer Slide Time: 25:53)

The Attack Procedure

Fault Model	Frequency range for both original and redundant rounds (MHz)
SBU	125.3-125.4
SBDBU	125.6-125.7
SBTBU	126.0-126.1
SBQBU	126.3-126.4

Fault Distribution

Distinguishers used :
 Hamming Distance (HD)
 Squared Euclidean Imbalance (SEI)

Make a key hypothesis k and evaluate the distinguishers
Correct hypothesis gives minimum and maximum values respectively

$$H(k) = \sum_{i=1}^{N_C} \sum_{j=1}^{i-1} HD(S^{tr}_{k, f_i}, S^{tr}_{k, f_j})$$

$$S(k) = \sum_{i=1}^{N_C} \sum_{\delta=0}^{255} \left(\frac{\# \{b \mid S^{tr}_{k, f_i}[b] = \delta\}}{N_C} - \frac{1}{256} \right)^2$$

So, then we basically do the same attack. So, remember that we basically pre characterize the device we kind of know that if the device is in this particular frequency

range or operated in this particular frequency range, then with more probability we get faults which are of the single bit upset nature. And likewise there are other fault characterizations also shown here in this table. And then we basically target that this particular implementation in both the original and the redundant rounds.

Remember these are time redundancy which has been implemented. So, now, we basically kind of inject a fault in both situations and, but whenever we are injecting right we are basically ensuring that the operating clock is in this region of one say 125.32 125.4 Mega Hertz. So, you can see that I am operate trying to operate in a region which I know that is more probable to create single bit upsets and then basically I start guessing the key, and as I said that I would like to minimize the humming distance or I can also use another alternative parameter which is called as the squared Euclidean imbalance. So, I would try to kind of maximize this parameter.

Because you can see that you know like if the guess is random then this would be something like 1 by 256 and therefore, this would be pretty much going to 0, but on the other hand right if my guess is correct then this I will get a you know like a non zero value here and if I add up all those non zero squares right then I should get a larger value. So, therefore, in this case the correct key would minimize this parameter of H_k , but in this case the correct key would kind of maximize the parameter S_k . So, you can actually use both the matrix or both the distinguishes together just to increase your confidence.

(Refer Slide Time: 27:30)

Simulations-1

- **Identical faults** introduced into both original and redundant rounds
- Target byte chosen at random
 - Same fault for original and redundant computations
 - Each fault injection yields a *useful ciphertext*
- Attacks simulated on rounds 8 and 9
- Performed separately for each fault model

Number of ciphertexts required to guess the AES key with 99% accuracy

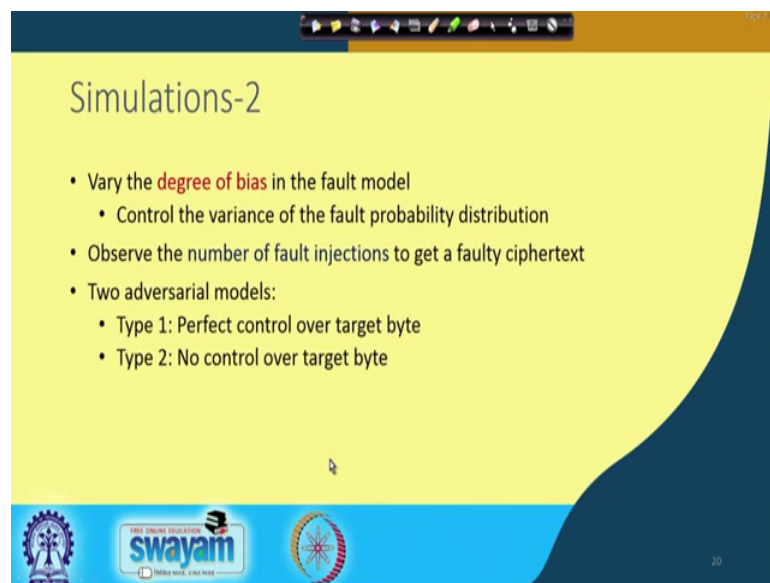
Simulation results

Round	Fault Model	N_C
8	SBU	320-340
	SBDBU	580-600
	SBTBU	1000-1040
	SBQBU	1900-2000
9	SBU	288-320
	SBDBU	608-640
	SBTBU	832-880
	SBQBU	1360-1440

swayam

So, then what we observe here is that we kind of do a simulation. So, in the simulation right we are basically assuming that an identical fault as we introduced into both the original and the redundant round, this is not a non actual hardware. And we observe that typically these are the number of faults that you require to get the AES key extracted. So, you can see that the fault number of faults here is much more in that in the classical DFA, but you can assume that here there are certain advantages which we did not have in the case of DFA.

(Refer Slide Time: 27:59)



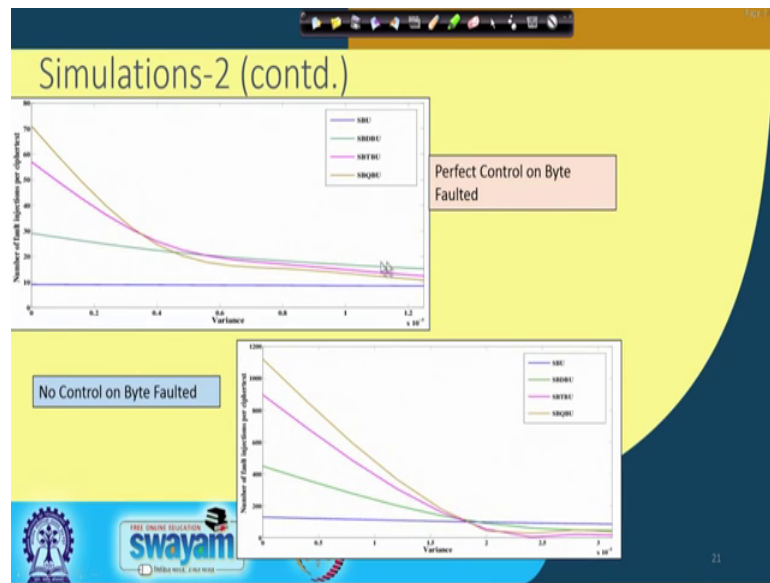
Simulations-2

- Vary the **degree of bias** in the fault model
 - Control the variance of the fault probability distribution
- Observe the number of fault injections to get a faulty ciphertext
- Two adversarial models:
 - Type 1: Perfect control over target byte
 - Type 2: No control over target byte

swamyam

So, likewise right you can actually' so we also need you know like a simulation 2 where you know like we are basically varying the degree of bias by you know like controlling the variance and then right we basically plot them.

(Refer Slide Time: 28:11)



And here are some plots to observe here you can see that if I increase the number of variance in both the cases right whether it is a scenario 1 or scenario 2 where you have got perfect control on the byte fault or you do not have perfect control on the byte fault; in both the cases the number of faults is reducing because as we have discussed right that if you increase the variance than the probability of successfully defeating this countermeasure is more and therefore, you need lesser and lesser number of observations.

(Refer Slide Time: 28:38)

Experimental Results

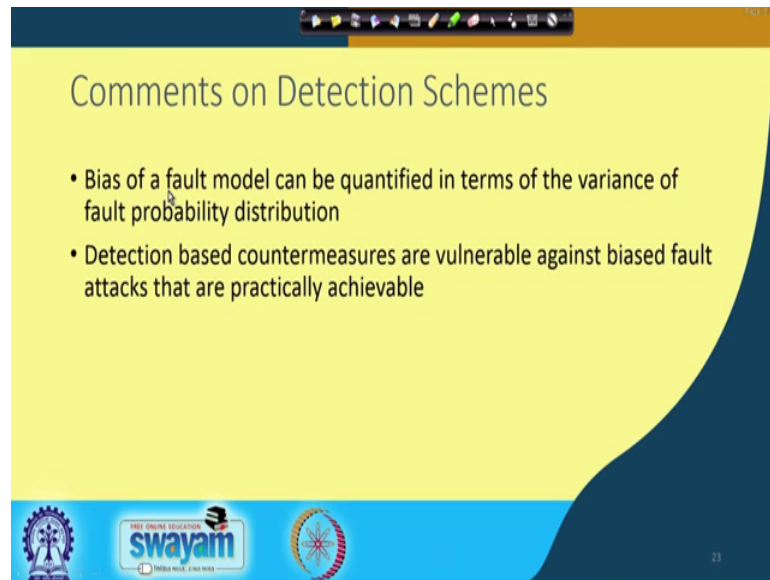
Useful ciphertexts

Total Fault injections

Round	Fault Model	Fault Variance		N_C	N_F (simulation)		N_F (experimental)	
		Type-1	Type-2		Type-1	Type-2	Type-1	Type-2
8	SBU	9.5×10^{-2}	3.6×10^{-3}	304.75	340.48	647.52	387.67	687.91
	SBDBU	1.4×10^{-2}	9.2×10^{-4}	625.12	1456.25	1506.25	1448.45	1652.30
	SBTBU	9.7×10^{-3}	4.9×10^{-4}	1020.49	1815.60	2315.40	1974.86	2395.83
	SBQBU	3.2×10^{-3}	5.9×10^{-5}	1878.55	7868.82	28038.54	8003.14	30201.41
9	SBU	9.2×10^{-2}	3.5×10^{-3}	304.24	385.88	603.11	387.98	632.71
	SBDBU	8.8×10^{-2}	7.9×10^{-4}	624.65	641.18	1487.36	647.82	1556.69
	SBTBU	8.1×10^{-2}	6.7×10^{-4}	832.32	873.56	2054.00	878.23	2489.25
	SBQBU	7.5×10^{-2}	3.5×10^{-5}	1328.22	1788.84	17239.10	1809.25	20145.66

In fact, you can observe this also on validate this on actual data. So, this particular column shows that what happens, what happens when you actually do it on a real life implementation with fault countermeasure using time redundancy, and you can see that the observations match pretty much with your theoretical simulations.

(Refer Slide Time: 28:59)



Comments on Detection Schemes

- Bias of a fault model can be quantified in terms of the variance of fault probability distribution
- Detection based countermeasures are vulnerable against biased fault attacks that are practically achievable

swayam
FREE ONLINE EDUCATION
INDIA WISE, FUTURE READY

23

So, therefore, right I mean the comments just to conclude right some of the comments would here would be that, the bias of a fault model can be quantified in terms of the variance of the fault probability distribution, and detection based countermeasures are vulnerable against biased fault attacks that are practically achievable.

(Refer Slide Time: 29:16)

• Fault Tolerance for DFA needs to be revisited?

Cover **all of the essential**
or
almost all???

The slide features a yellow background with a dark blue curved border on the right. At the top, there is a navigation bar with various icons. Below the text, there is an image of Superman in his iconic blue and red suit, with a glowing green 'S' on his chest. At the bottom of the slide, there are logos for 'swayam' (Free Online Education) and 'INDIA WIDE, 100% WIDE'. A small video inset of a man speaking is visible in the bottom right corner.

So, the question is right what do you really want to do? You will basically want to cover all of the essential what we do in classical fault tolerance or do you want to really cover almost all. So, therefore, in a classical fault tolerance we basically do almost all that is we kind of try to pretty much covered almost all, but what may happen is that, what we probably need to do here is that we probably need to cover all of the essential. Because you know like in faulted; in cryptography or security right even if there is a small window of you know like of an attack and the attacker can actually exploit that window and can you know like collapse the entire security of your system.

(Refer Slide Time: 29:50)

Summary: Types of Fault Attacks

- Differential Fault Analysis (DFA)
 - Induce a fault
 - Observe the Difference of the correct and faulty pairs
 - Derive equations to obtain the key
- Differential Fault Intensity Attack (DFIA)
 - Obtain non-uniform faults (biased faults) through non-expensive techniques
 - Perform Side Channel Analysis like power analysis to exploit the bias

The slide features a yellow background with a dark blue curved border on the right. At the top, there is a navigation bar with various icons. Below the title, there is a list of two types of fault attacks: Differential Fault Analysis (DFA) and Differential Fault Intensity Attack (DFIA). At the bottom of the slide, there are logos for 'swayam' (Free Online Education) and 'INDIA WIDE, 100% WIDE'. A small video inset of a man speaking is visible in the bottom right corner.

So, to summarize we have discussed about DFA in the previous classes, where we basically induce a fault observe the difference of the correct and faulty pairs derive equations to obtain the key.

On the other hand what we discussed in today's class is what is called as DFIA or Differential Fault Intensity Attack. So, here we obtain non uniform-faults or biased faults you can see that, it is a difference from the DFA where we assume random faults through non-expensive techniques and then we performed side channel analysis like power analysis to exploit the bias.

(Refer Slide Time: 30:20)

Conclusions

- Detection based schemes are a popular strategy to prevent fault attacks.
- CED schemes are efficient fault tolerance strategy.
- Parity based schemes are useful for making AES-like cryptosystems resistant against fault attacks
- Time Redundancy Countermeasures are vulnerable to DFIA

So, to conclude detection based schemes are a popular strategy to prevent fault attacks. CED schemes or conquer a data detection schemes are efficient fault tolerance strategy. Parity based schemes are useful for making AES-like cryptosystems resistant against fault attacks, and time redundancy countermeasures are vulnerable to DFIA.

(Refer Slide Time: 30:41)

References:

- Debdeep Mukhopadhyay and Rajat Subhra Chakraborty, *Hardware Security: Design, Threats and Safeguards*, CRC Press

HARDWARE SECURITY
Design, Threats, and Safeguards
Debdeep Mukhopadhyay
Rajat Subhra Chakraborty

swamyam
FREE ONLINE EDUCATION
BIRSA INSTITUTE OF TECHNOLOGY

So, we still need to kind of figure out what to do, and the references that we have followed here from for this part is shown here and.

Thank you for your attention.