

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

**Lecture – 56**  
**Microarchitectural Attacks: Part 2 Cache Timing Attacks on Block Ciphers**  
**(Contd.)**

So, welcome back to this class on Hardware Security. So, we shall be continuing our discussion on cache timing attacks. In particular we shall be trying to talk about cache timing attacks on small table ciphers and we shall see them on impacts of Paralyzation and Out-of-Order Loading.

(Refer Slide Time: 00:31)



In order to perform this because last day we saw that cache timing attacks on a cipher implementations which small tables seems to be more difficult than doing on block ciphers implemented with larger tables.

And then we shall try to talk about a case study with an example of cipher which is called as Clefia.

(Refer Slide Time: 00:50)

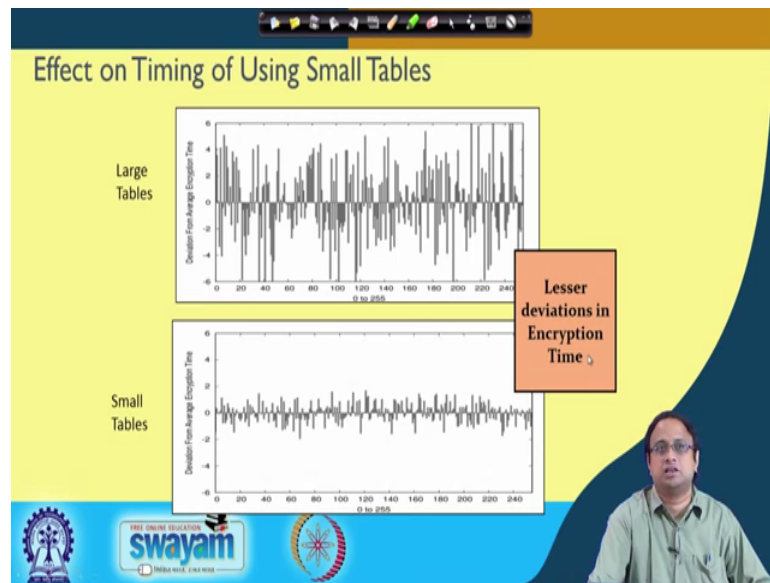


So, therefore, right let us take a just recapitulate that the fact that, we saw that when there is a cipher which is implemented with a small table, then the variation of the number of misses is gone because we essentially have got a fixed number of misses ok. So, what is the impact of this on the Osvik's Attack? So, again let us take again recapitulate the Osvik's attack, but now where a cipher has been implemented with a small table like AES has well implemented say with 256 values ok.

So, therefore, right what will happen here is that. So, I am just storing the basically the S-Box as the table rather than the entire round as the table. So, the S-Box become smaller, now the table becomes smaller. So, here what happens is that now we basically again given access to the spy. So, therefore, the spy again fills up the cache memory with garbage data now the access comes back to AES. So, the AES now starts doing encryption. So, remember now I have a got a small table so, therefore, after few execution the entire thing is removed from the cache memory.

So, now right if the spy comes back then what will happen is that, it will not basically get the nice distribution. So, in remember in the previous case we got like few hits and few misses. So, if basically aids because it gives you a nice distribution of cache hits and cache misses. But now right everything is becomes a miss and that essentially becomes an issue.

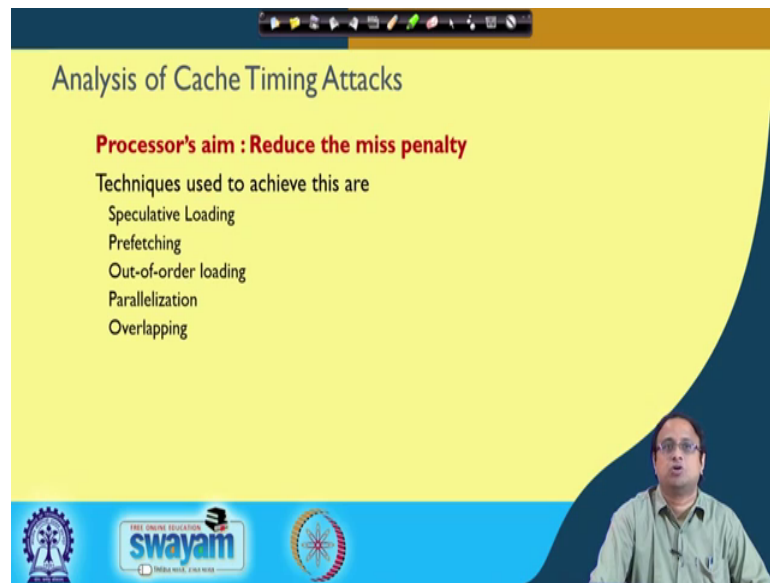
(Refer Slide Time: 02:17)



So, now, right I mean; so, the even in context of the effect of timing using small tables. So, if you see that when you have got large tables, then the deviation from the average is also very large. So, you see that there is a large variation because the number of misses across various executions are deferring ok. Like when I am doing say the first encryption, I get large; I get a certain number of misses.

But in the next encryption I can get a completely different number of misses and that means, the timing would vary. On the other end here now the number of misses becomes a constant with small tables and therefore, right you see that there are lesser deviations in time that we expect. For example, right here is an example where the cipher has been implemented with small tables and you can see that the deviation from average is really really small ok. So, the question is can you still exploit them, can you still kind of you know like do an attack with them or this deviation is really like a random deviation which cannot be exploited.

(Refer Slide Time: 03:10)



**Analysis of Cache Timing Attacks**

**Processor's aim : Reduce the miss penalty**

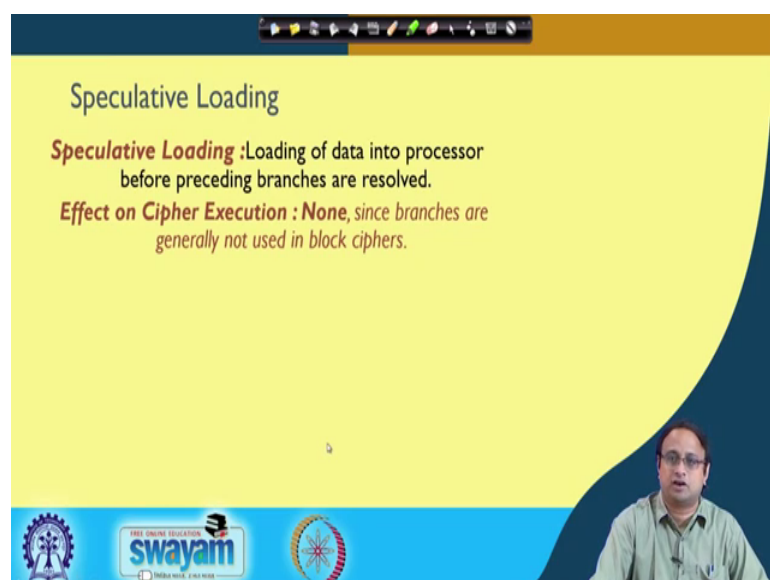
Techniques used to achieve this are

- Speculative Loading
- Prefetching
- Out-of-order loading
- Parallelization
- Overlapping

The slide features a yellow background with a dark blue curved shape on the right. At the bottom, there are logos for Swamyam and other educational institutions, along with a small video inset of a man in a green shirt.

So we will see that you know like using a certain kind of feature in architecture, you can rationalize and even that small timing right is not random and essentially has got a relationship with the secret key. So, therefore, right; so, let us take a look back into you know like some of the architectural techniques, for example, as I as we already discussed that the processors fundamental aim is to reduce the miss penalty and enhance performance. So, because of that there are several techniques which has been proposed and adopted in architectures; for example, speculative loading, prefetching, out-of-order loading, parallelization's, overlapping ok.

(Refer Slide Time: 03:48)



**Speculative Loading**

**Speculative Loading** :Loading of data into processor before preceding branches are resolved.

**Effect on Cipher Execution** : *None, since branches are generally not used in block ciphers.*

The slide features a yellow background with a dark blue curved shape on the right. At the bottom, there are logos for Swamyam and other educational institutions, along with a small video inset of a man in a green shirt.

So, therefore, just to take a look one by one at them. In speculative loading you load a data into processor before proceeding branches are resolved ok. So, therefore, right what is the effect of this on a block cipher kind of execution? Probably none because you know like in branches they are not generally used in block ciphers ok. So, there are no branches as such so; that means, like in particular I am talking about speculations with respect to say branch branches ok. So, therefore, right it does not seem that there would be a direct leakage because of this kind of architectural features.

(Refer Slide Time: 04:25)

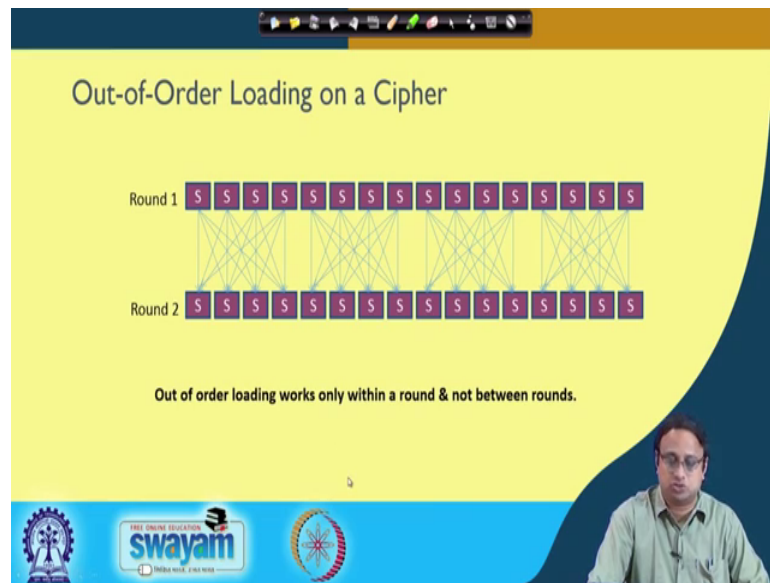
The slide is titled "Out-of-Order Loading" and contains the text "Data accessed in an order not strictly specified by the program." It features two columns: "Program" and "Execution".

| Program        | Execution      |
|----------------|----------------|
| LOAD AX, #1000 | LOAD DX, #4000 |
| LOAD BX, #2000 | LOAD BX, #2000 |
| LOAD CX, #3000 | LOAD AX, #1000 |
| LOAD DX, #4000 | LOAD CX, #3000 |

The slide also includes a Swayam logo and a small video inset of a presenter in the bottom right corner.

On the other hand right what is out of order loading? So, it means that data accessed in an order it is like if the program is in a specific order, the execution may be in a different order ok so, depending upon the fact whether there any two dependencies across instructions.

(Refer Slide Time: 04:40)



So now, let us take an example of the impact of out of over load order loading where a block cipher has been implemented and essentially we are considering a software code for a block cipher. So, we know that in round one for example. So, if I take the example of AES without loss of generality, there are several S Box table axis that you are doing and all of them are independent ok.

So, next time when you are doing a round encryption again you are doing a bunch of S-Box axis, but remember that in a round the S-Boxes are independent there are no dependencies, but across round there are dependences ok. That means, if you want to do out-of-order loading inside a round you can do out-of-order loading, but across round you cannot do out-of-order loading because there is aberrance ok. So, therefore, right out-of-order loading works only within a round and apparently not between the rounds. So, that is one of the observations.

(Refer Slide Time: 05:31)

**Parallelization & Overlapping**

**Parallelization** : Two or more cache misses are serviced simultaneously.

**Overlapping** : Memory reads are pipelined.

**Effect on Cipher Execution** :

Consider a 5 round block cipher with 4 misses

**Time(E1) < Time(E2)**

|    |          |    |          |
|----|----------|----|----------|
| E1 | 2 misses | E2 | 1 miss   |
|    | 1 miss   |    | 1 miss   |
|    | 1 miss   |    | 1 miss   |
|    | 0 misses |    | 0 misses |
|    | 0 misses |    | 1 miss   |

Logos for Swamyam and other institutions are visible at the bottom of the slide.

So now, let us try to look at another feature which is due to the you know like the support of you know like which modern day computer architectures provide to handle two or more cache misses in parallel ok. So, what our modern day architecture support right is that if there are two or more cache misses and there are no dependence between them, then you can essentially pipeline their memory reads ok; that means, you can essentially perform one of them without waiting for the other one to complete ok. And these are very interesting feature and very important feature which enhances the overall you know like overall performance of the cache memory.

So, therefore, right let us consider a scenario or impact of this on cipher execution by taking a very simple example of a 5 round block cipher; hypothetical 5 round block cipher where there are 4 cache misses ok. So, even in that in the first execution you have got the 4 cache misses like 2, 1, 1. So, that mean the distribution is like in the first round there are 2 cache misses, in a second round there is 1 miss, in the third round there is 1 miss ok. So, let us consider another execution and remember that we are considering ciphers with small tables and therefore, the number of cache misses remains constant again 4, but the distribution is different. So, we get like one 1, 1, 1 and then 0 and 1 ok.

So, the question is right what would be the timing required for E1 and E2. So, we saw that in the previous case if I just do not consider all these kind of tricks like you know like the out-of-order execution or the out-of-order loading, overlapping, parallelization if

I just kind of not consider them, then I would expect that the execution time for E1 and E2 would be roughly the same because the number of cache misses is the same, but there is a difference. For example, here we see that the 2 cache misses that are happening over here essentially can be handled in parallel because there are no dependencies across them ok.

On the other hand here the 4 cache misses right since they are happening one after the other they cannot be handled in parallel. So, therefore, right although the cache misses here and the cache misses here are same because of these features, I would expect the time of E1 would be less than the time of E2 ok. There will be a small timing difference, but the timing difference essentially has got aberrance on the distribution of cache misses; that means, although the total number of cache misses are still constant, the distribution of cache misses are varying and that essentially can potentially leak information ok.

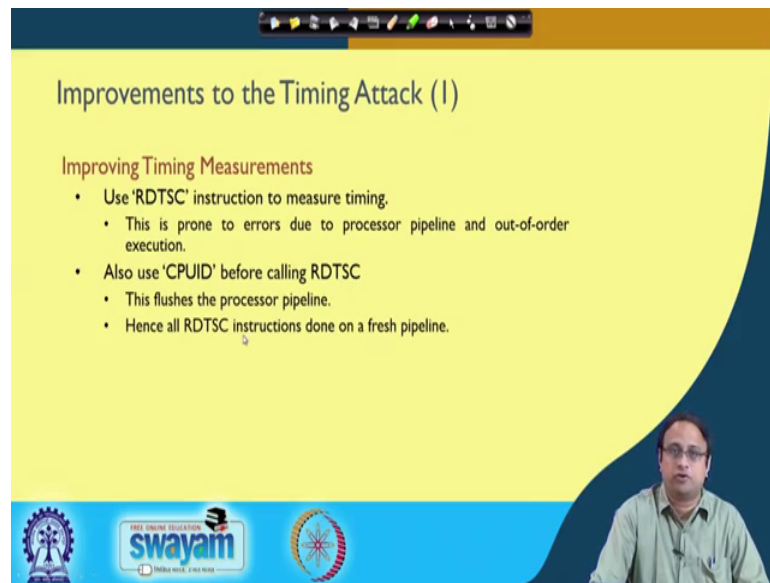
(Refer Slide Time: 08:04)

The slide is titled "Extending Cache Timing Attack to Work on Small Table Ciphers". It lists two challenges: "1. The deviation in encryption time is too less." and "2. Timing measurements need to be more accurate." Below the text is a line graph showing "Deviation From Average Encryption Time" on the y-axis (ranging from -6 to 6) against "ID No 0000" on the x-axis (ranging from 0 to 240). The graph shows a highly volatile signal with many peaks and troughs. At the bottom of the slide, there are logos for "swayam" and "INDIA WIDE EDUCATION" along with a small video feed of a man in a green shirt.

So, therefore, right if I want to extend cache timing attacks to work on ciphers with small tables; that means, right although the deviation in encryption time is too less right, but it is still not random ok. And essentially can be potentially exploited, but of course, right we also must keep in mind that the timing measurements needs to be much more accurate.



(Refer Slide Time: 08:27)



Improvements to the Timing Attack (1)

Improving Timing Measurements

- Use 'RDTSC' instruction to measure timing.
  - This is prone to errors due to processor pipeline and out-of-order execution.
- Also use 'CPUID' before calling RDTSC
  - This flushes the processor pipeline.
  - Hence all RDTSC instructions done on a fresh pipeline.

swayam  
INDIA WISE, LEAD WISE

So, one of the improvements that we would like to do here for is that, rather than using a standard way of measuring time we would use this time stamp counter which is shown as 'RDTSC' instruction to measure timing ok.

So, this is; but this is also you know like prone to errors. So, the processor pipeline an out of order execution. So, one of the things that we can do is that we will be using a specific instruction called a 'CPUID' before calling RDTSC, this would flush the processor pipeline and therefore, all RDTSC instructions will be measured and done on a fresh pipeline ok. So, it is kind of reduces the amount of non determinacy ok. And actually there is another further instruction which you can also use nowadays which is like which is called as RDTSCP ok. And you can probably you know like improve the quality of your timing measurements even more ok.

(Refer Slide Time: 09:20)

The slide features a large diagram of a Feistel network on the left, showing multiple rounds of encryption with subkeys  $K_1, K_2, K_3, K_4$  and round functions  $F_0$  and  $F_1$ . On the right, two smaller diagrams illustrate the internal structure of  $F_1$  and  $F_0$ .  $F_1$  consists of two S-boxes ( $S_1$  and  $S_0$ ) followed by a linear transformation  $M_1$ .  $F_0$  consists of two S-boxes ( $S_0$  and  $S_1$ ) followed by a linear transformation  $M_0$ . Below these diagrams, the following text is displayed:

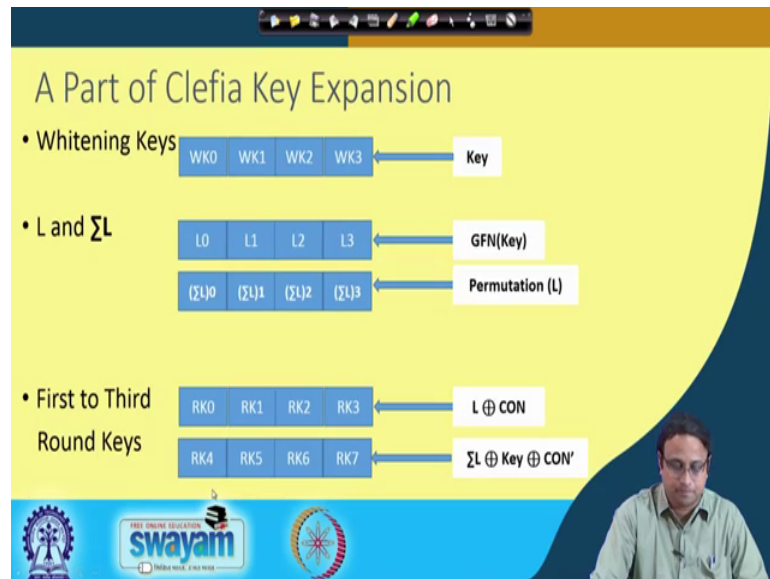
128 bit block cipher from Sony.  
Generalized Feistel Structure  
Number of rounds : 18  
Whitening Keys added at the beginning and end.  
Attacking Clefia requires finding any set of 4 round keys.  
RK0, RK1, RK2, RK3

So, now let us take a look or you know like see a case study of a block cipher called Clefia which was designed originally by Sony for protecting their digital copyrights. It is a 128 bit cipher and a block cipher which has been implemented with small S-Boxes so. So, therefore, right in this cipher this is an example or depiction of the cipher, this cipher essentially has got several rounds of encryption you know like around 18 rounds of encryption and this also like operates on 128 bits of block cipher at least in one of the variants.

And then if you observe right there are two non-linear. So, this is example of a Feistel cipher, more accurately it is a generalized Feistel cipher where there are two non-linear lookups; one of them is  $F_0$  and the other one is  $F_1$  ok. So, here is how the  $F_0$  looks like? There are two S-Boxes internally like  $S_1$  and  $S_0$  which are shown over here and followed by a linear transformation shown as  $M_1$  ok.

Similarly  $F_1$  is also the same S-Boxes  $S_0$  and  $S_1$ , but slightly the order has been changed followed by another linear transformation which is denoted as  $M_0$  ok. So, then what we do is there is another difference in the cipher compared to AES for example, this is an example of a cipher where there is an input and output whitening key; that means, you know like you have got another key which we essentially used to actually generate the round keys, but you basically exhort them at the beginning and also at the end ok.

(Refer Slide Time: 10:58)



So therefore right we will probably take a quick look at least a part of the Clefia key expansion. So, you have got this whitening key is shown as WK0, WK1, WK2 and WK3. So, what happens is that you basically develop an internal variable denoted as L and sigma L and there is a function called as GFN which I am not going in details. Where you basically create this L0, L1, L2 and L3 by applying this GFN function on the secret key and then you basically again apply this sigma L. So, this is basically a result of the permutation which you do and then you obtain the individual round keys by you know like applying certain operations of XORing this l with a value which is CON ok.

And then you obtain the first to third round keys and these round keys are denoted as RK0, RK1, RK2 and RK3 ok. So, our objective right would be obtain the, you know like the correspond if you are able to again you know like obtain the entire round key, then you essentially are again to again able to obtain the original primary key and that kind of breaks the security of Clefia. So, there are you know like bunch of other round keys for example, like you have got RK0, RK1, RK2 and RK3 you have got RK4, RK5, RK6 and RK7. So, our objective will be to recover the entire round key and that breaks the security of this cipher.

(Refer Slide Time: 12:23)

### Improvements to the Timing Attack (2)

Vary only those bytes which go to the same sbox as P0.

P0 = 00 - FF

Constant

Random

AES

So, therefore, right in order to adopt the timing attack on Clefia because remember it is a Feistel cipher we do another modification. So, here right here remember that I had only one branch which goes to this S-Box, but in Clefia which is a Feistel cipher it may happen that to an S-Box. There are different branches from different secret keys which goes into that S-Box and that makes the attack little bit more complicated because you have to vary only those bytes which go to the same S-Box as and your target S box whereas, the other parts you have to keep constant.

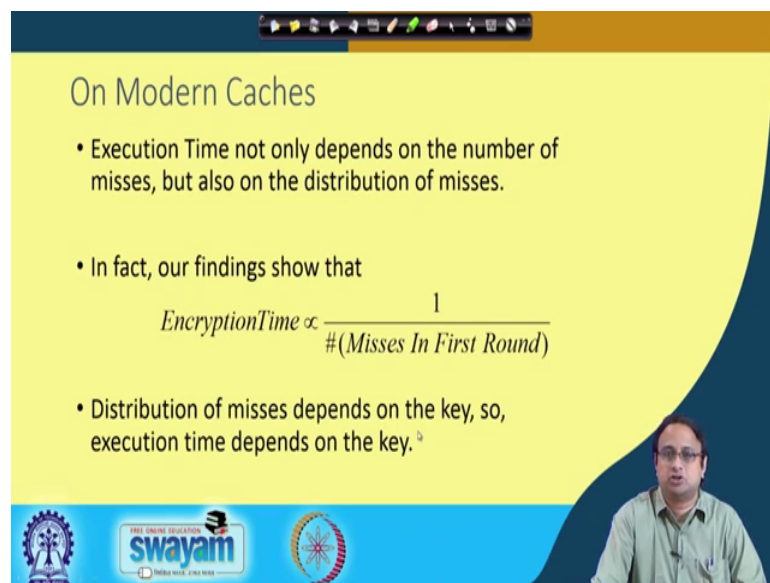
(Refer Slide Time: 12:57)

### Effect of Out-of-Order Loading on Clefia

Out of order loading works only within a round & not between rounds.

And also right just to remember that we are considering the effect of out of order loading on Clefia. So, that effect should still be there because again there are non-linear lookups like F0 and F1 where there are internal access to the S0 and S1 and remember that again inside one round that is suppose in round 1, the accesses to S0 S1 and S0 S1 here in F1 they are all independent and therefore, they can be serviced in parallel whereas, if you go across rounds they cannot be serviced in parallel because there is this dependency. So, therefore, out of order loading works only within a round and not between the rounds.

(Refer Slide Time: 13:34)



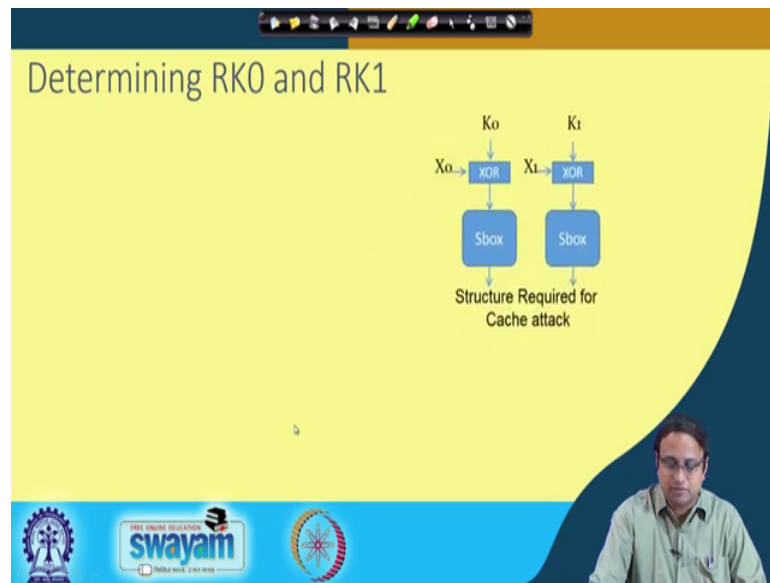
On Modern Caches

- Execution Time not only depends on the number of misses, but also on the distribution of misses.
- In fact, our findings show that
$$\text{EncryptionTime} \propto \frac{1}{\#(\text{Misses In First Round})}$$
- Distribution of misses depends on the key, so, execution time depends on the key. <sup>2</sup>

swayam

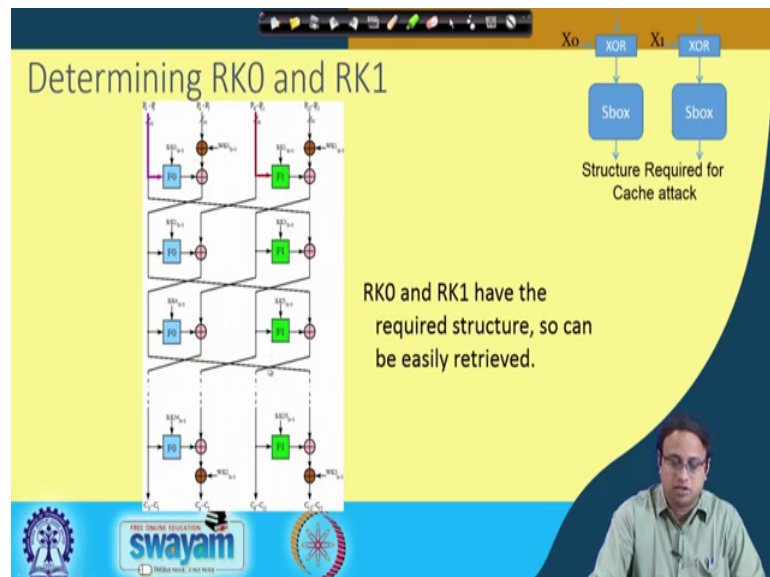
So, on modern caches the execution time not only depends on the number of misses, but also on the distribution of misses. So, this is the fact that we would like to exploit. In fact, our findings show that the encryption time is directly proportional to one by number of misses in the first round ok. So, I leave it to you as an exercise to think that why this is true ok. Remember that we are talking about small table implementations where the total number of misses is a constant. So, therefore, right what is important is that the distribution of misses depends on the key so, execution time depends on the key.

(Refer Slide Time: 14:10)



So, now let us see a step by step process of recovering the round keys ok. The first part is very easy. So, this is the structure which is required for cache attack. So, again this part is quite similar to what we have seen in the context of AES and that is even true in the case of Clefia here.

(Refer Slide Time: 14:26).



So, what we do is we basically target F0 we target F1 and we recover RK0 and RK1. So, this part is exactly similar we basically again do a timing profile analysis and again do a correlation to obtain the value of RK0 and RK1 in this fashion.

(Refer Slide Time: 14:43)

The slide features a yellow background with a blue header and footer. The title 'RK2 and RK3 not directly possible' is in the top left. A diagram on the left shows a cipher structure with four rounds, each containing two S-boxes and XOR operations. The right side has a smaller diagram showing two S-boxes with XOR operations and inputs  $X_0$  and  $X_1$ . A list of bullet points is on the right, and a small video inset of a speaker is in the bottom right corner.

RK2 and RK3 not directly possible

- Not possible to determine RK2 and RK3
- But  $RK2 \oplus WK0$  and  $RK3 \oplus WK1$  using previously determined  $RK0$  and  $RK1$ .

Now, we want to know whether we can recover the other parts of the key. So, now, you see that this becomes more complicated because it is not possible to determine RK2 and RK3 directly because there is an unknown whitening key WK0 and WK1 which I do not know ok. So, what I can obtain is basically the XOR of WK0 and RK2. Remember right if you go back to the structure this gets XORed here and therefore, right I can obtain the XOR of RK2 and WK0 and I can obtain the XOR of RK3 and WK1 ok.

So, therefore, right I mean RK3 and WK1. So, therefore, I can obtain these XORs and that is possible. So, therefore, RK2 XOR with WK0 and RK2 so, this we can obtain and you know like we can obtain the value of this using previous determine RK0 and RK1. So, RK0 and RK1 has already been previously deciphered and therefore, we can use that knowledge because you know like since I know the value of RK0 and RK1, I can now keep this part you know like I can now vary this part, but since I know RK0 and RK1 I pretty much know what is the input which is coming here and I know what is the input which is coming here ok.

So, I have got control till this points because I have already deciphered the value of RK0 and RK1. So, then I again repeat a process similar to the previous attack and I obtain the XOR between this key and this key and this key and this whitening key.



(Refer Slide Time: 16:12)

The slide is titled "Determining RK4 and RK5". It features a complex diagram of a cipher's internal structure on the left, showing multiple rounds of computation with various operations like XOR, AND, and OR. On the right, there is a smaller diagram showing two parallel paths starting from inputs  $X_0$  and  $X_1$ , each passing through an XOR operation with keys  $K_0$  and  $K_1$  respectively, followed by an S-box operation. Below the diagrams, a list of steps is provided:

- RK4 and RK5 can be determined
- Using RK0 and RK1
- And  $RK2 \oplus WK0$  and  $RK2 \oplus WK0$

The slide also includes logos for "swayam" and "INDIA WISE, LEAD WISE" at the bottom left, and a small video inset of a person in the bottom right corner.

So, now we want to know whether we can obtain further. So, therefore, RK4 and RK5 is the next thing that we target and it is interesting to see that RK4 you can recover because this is the branch which is affects RK4 and there is no whitening key in this branch ok.

So, now you have already recovered RK0 you have recovered the XOR of RK2 and WK0. So, you can again have a control at this point because of this and therefore, right you can again by a similar analysis obtain the value of RK4 and RK5. RK4 and RK5 can be determined using RK0 and RK1 and the XOR of RK2 and WK0 and the XOR of RK3 and WK1 ok. So, this is again typo this should this should be RK3 XOR with WK0.



(Refer Slide Time: 17:04)

Determining RK2 and RK3

- $L_{(0-63)} = (RK0 \mid RK1) \oplus (CON)$
- $(\Sigma L)_{(0-56)} = L_{(7-63)}$
- $WK0 \mid WK1_{(0-24)} = (\Sigma L)_{(0-56)} \oplus (RK4 \mid RK5) \oplus (CON')$
- Using this the whole of RK2 and 25 bits of RK3 can be found.
  - We therefore have RK0, RK1, RK2, and 25 bits of RK3.

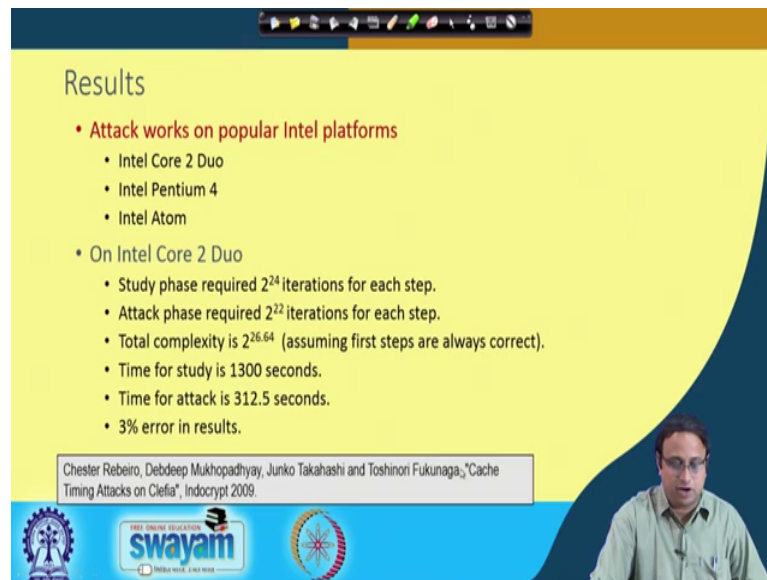
The slide features a yellow background with a blue and orange header. At the bottom, there are logos for 'swayam' and 'INDIA WISE, LEAD WISE' along with a small video inset of a man in a green shirt.

So, let me just correct it. So, this is the XOR of RK3 and WK; so, this is the XOR of RK3 and WK1 ok. So, therefore, right what we do now is we go ahead and we basically right see that whether we can recover the remaining parts of the key in a similar fashion and so now, we determine RK1 to determine RK2 and RK3, but for that we have to again resort to the key scheduling algorithm. So, remember that the L right is essentially obtained from the round keys and this is the function for doing that. We already know the value of RK0 and RK1 CON is a constant. So, we know the value of L ok.

And now, therefore, we can again obtain the value of sigma L because this function is a public function and remember that again by seeing the equations apparently that is you know, you can you know the value of sigma L, you know the value of RK4 and RK5 and again there is a constant which if we apparently if we XOR and if you see the key scheduling equation of Clefia, then you will be able to obtain the entire value of WK0 and 25 bits of WK1 ok.

Again if you observe the key scheduling algorithm right from these known values like the entire WK0 and 25 bits of WK1 you can obtain the whole of RK2 and 25 bits of RK3 ok. So, therefore, right we have obtained RK0 and RK1, RK2 and 25 bits of RK3 the remaining bits of RK3 right we can essentially just do a brute force because that is essentially pretty small.

(Refer Slide Time: 18:42)



**Results**

- **Attack works on popular Intel platforms**
  - Intel Core 2 Duo
  - Intel Pentium 4
  - Intel Atom
- **On Intel Core 2 Duo**
  - Study phase required  $2^{24}$  iterations for each step.
  - Attack phase required  $2^{22}$  iterations for each step.
  - Total complexity is  $2^{46.64}$  (assuming first steps are always correct).
  - Time for study is 1300 seconds.
  - Time for attack is 312.5 seconds.
  - 3% error in results.

Chester Rebeiro, Debdeep Mukhopadhyay, Junko Takahashi and Toshinori Fukunaga, "Cache Timing Attacks on Clefia", Indocrypt 2009.

swayam

So, therefore, right now if you again repeat this attack this is the result of this attack on popular Intel platforms like Intel Core 2 Duo, Intel Pentium 4, Intel atom and here is the summary of the attack you can see that the study phase requires around 2 power of 24 iterations for each step. Remember that you have to do; when you are doing the study right you have to basically repeat the process several times. So, you get a accurate measurement of the average timing and then you do your correlation. So, again in the attack phase when you are correlating you are doing around slightly lesser, but still quite large like 2 power of 22 iterations for each step.

And the total complexity of this attack is 2 power of 26.64 assuming that the first steps are always correct ok; that means, I am assuming that the steps that we are doing one by one are correct. So, overall like if you launch this attack apparently the study time; that means, the template 300 time takes around 1000 seconds more accurately 1300 seconds. But the time for the actual attack is pretty small; it takes around 312 seconds to do the attack. And there is a small error percentage which is around 3 percent interestingly right what happens is that if there is a particular error in retrieving a key byte, then when you repeat the experiment you will see often that that bit, that byte creates an error.

So, you kind of by you know like combining the two results you know that which byte you are not is kind of changing. So, you know that this particular byte in the key are not able to retrieve whereas, the other bytes are you are more confident about them. So,

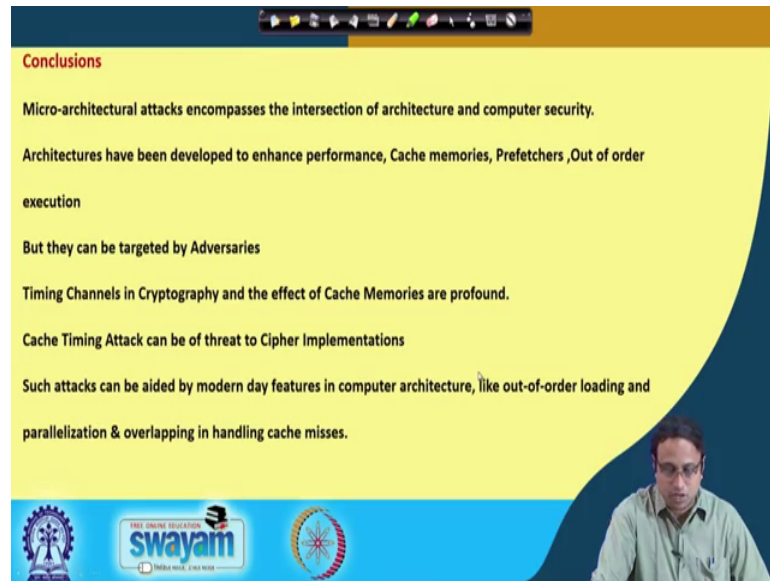
therefore, you can do a brute force search in that key byte ok. So, this is the sort of you can find this reference from this paper published in Indocrypt, but it shows that overall how you can do basically it is a summary of the attack that we just now discussed.

(Refer Slide Time: 20:18)

| Key Byte                            | Correct Key | Obtained Correlation results (with correlation value)                   |
|-------------------------------------|-------------|---|
| RK0 <sub>0</sub>                    | 0a          | 0a(854.6), 0b(469.7), 5f(568.3), 20(357.3), cf(263.7) ...               |
| RK0 <sub>0</sub>                    | 96          | <b>96(1853.4)</b> , 7b(438.0), bc(437.5), 4a(366.7), ee(361.8) ...      |
| RK0 <sub>2</sub>                    | c1          | c1(1942.1), 93(672.7), 98(598.3), 0(573.2), 24(559.5) ...               |
| RK0 <sub>3</sub>                    | 08          | <b>08(1680.3)</b> , 23(415.9), 9e(414.1), 6e(398.9), 90(375.9) ...      |
| RK1 <sub>0</sub>                    | ac          | ac(4077.6), c1(853.4), 11(843.3), 7c(650.9), 71(639.2) ...              |
| RK1 <sub>1</sub>                    | b0          | <b>b0(3089.8)</b> , 73(740.8), 07(716.7), f7(677.1), 01(658.1) ...      |
| RK1 <sub>2</sub>                    | 7a          | 7a(5721.0), 0a(1539.1), 08(1230.2), 6f(967.9), 05(901.9) ...            |
| RK1 <sub>3</sub>                    | 79          | <b>79(5361.6)</b> , 0b(1802.0), 2b(1196.0), 9a(1106.6), 07(1007.9) ...  |
| RK2 <sub>0</sub> ⊕ WK0 <sub>0</sub> | 6e          | <b>6e(4194.0)</b> , 09(1536.2), 07(1491.3), 96(1257.9), 2f(1194.3) ...  |
| RK2 <sub>1</sub> ⊕ WK0 <sub>1</sub> | b1          | <b>b1(4344.0)</b> , 39(1197.5), 59(1056.8), 03(980.9), 0(926.9) ...     |
| RK2 <sub>2</sub> ⊕ WK0 <sub>2</sub> | 9f          | <b>9f(2662.0)</b> , d4(1327.9), 08(1071.1), 1b(1056.2), 80(1000.0) ...  |
| RK2 <sub>3</sub> ⊕ WK0 <sub>3</sub> | 61          | <b>61(6840.2)</b> , 0a(1783.8), 97(1587.3), 8c(1555.8), 87(1491.4) ...  |
| RK3 <sub>0</sub> ⊕ WK1 <sub>0</sub> | c3          | <b>c3(21042.8)</b> , 38(464.1), ea(4429.9), d3(3999.8), 01(3995.1) ...  |
| RK3 <sub>1</sub> ⊕ WK1 <sub>1</sub> | 85          | <b>85(24258.3)</b> , 54(8695.1), 83(8576.9), 3a(840.3), ee(8318.3) ...  |
| RK3 <sub>2</sub> ⊕ WK1 <sub>2</sub> | 2c          | <b>2c(27773.2)</b> , 3c(7131.3), 28(6804.1), 05(6263.3), b5(5906.3) ... |
| RK3 <sub>3</sub> ⊕ WK1 <sub>3</sub> | 4d          | <b>4d(32267.7)</b> , f2(9903.8), 33(9625.5), 24(8613.2), cf(8595.4) ... |
| RK4 <sub>0</sub>                    | 3f          | <b>3f(1321.7)</b> , 5e(535.2), 39(328.4), 83(302.9), 04(276.8) ...      |
| RK4 <sub>1</sub>                    | df          | <b>df(2066.6)</b> , e6(510.7), 09(463.6), ad(441.4), 5a(399.3) ...      |
| RK4 <sub>2</sub>                    | d7          | <b>d7(1367.1)</b> , 09(331.8), b5(322.7), bc(319.7), 39(313.6) ...      |
| RK4 <sub>3</sub>                    | 5f          | <b>5f(1330.7)</b> , cb(409.6), ae(392.4), 1e(373.3), ee(365.7) ...      |
| RK5 <sub>0</sub>                    | 66          | <b>66(5056.0)</b> , 4e(308.3), 01(294.7), b6(286.9), 05(270.5) ...      |
| RK5 <sub>1</sub>                    | 97          | <b>97(3579.2)</b> , e4(795.3), 54(794.1), 42(674.6), 4a(633.2) ...      |
| RK5 <sub>2</sub>                    | 2d          | <b>2d(6248.1)</b> , 5f(1313.0), 5d(1274.5), b3(1180.1), 38(1134.4) ...  |
| RK5 <sub>3</sub>                    | 4e          | <b>4e(6405.4)</b> , ec(1963.7), 8d(1173.4), 0(1147.6), 1a(1140.9) ...   |

So, here is a quick snapshot into one of the results, you can see that what we plot here this is the correct key and we plot the correlation of the correct key with also the wrong keys and you can find that the correlation of the wrong keys are almost half compared to the correlation of the correct keys or the correct key bytes and which shows that the exactly you know the entire 16 bit; 16 bytes of the key has been recovered in case of Clefia ok. So, that leads to a complete attack on Clefia using cached timing attacks.

(Refer Slide Time: 20:51)



**Conclusions**

Micro-architectural attacks encompasses the intersection of architecture and computer security.

Architectures have been developed to enhance performance, Cache memories, Prefetchers, Out of order execution

But they can be targeted by Adversaries

Timing Channels in Cryptography and the effect of Cache Memories are profound.

Cache Timing Attack can be of threat to Cipher Implementations

Such attacks can be aided by modern day features in computer architecture, like out-of-order loading and parallelization & overlapping in handling cache misses.

swayam  
INDIA WIDE, 24x7 WIDE

So, to summarize microarchitecture attacks encompasses the intersection of architecture and computer security, architectures have been developed to enhance performance and therefore, we have got cache memories, prefetchers, out of order execution, branch predictions ok, but we will see more in details in some of the future classes, but they can all be targeted by adversaries ok.

So, timing channels in cryptography and the effect of cache memories are profound and we have seen some of the attacks. In particular we have seen different kinds of cache attacks like cache trace attacks, cache access attacks and cache timing attacks. In particular cache timing attacks can be of great threat to cipher implementations because they can be remotely performed and we have seen that, such attacks can also be aided by modern day features in computer architecture like out of order loading and parallelization and overlapping in handling cache misses ok.

(Refer Slide Time: 21:54)

**References:**

- Discusses various timing attack algorithms in detail allowing readers to reconstruct the attack.*
- Presents the application of timing attacks on remote systems and cloud environments.*
- Examines information leakage models that would help quantify leakage in a covert timing channel!*

**Timing Channels in Cryptography**  
A Micro-Architectural Perspective  
Diederik Boenigk, Gokulakrishnan Venkatesh, Suresh Babu Babu  
Springer

swayam  
FREE ONLINE EDUCATION  
BETTER QUALITY. FASTER LEARNING.

So, in the next class right we shall continue on our discussions on micro architecture attacks, and I would also like to point out that one of the important references that I use in this particular part is this book called as Timing Channels in Cryptography published by Springer and you will find that there are several attack descriptions which are given in this textbook.

And so, thank you for your attention.