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

Lecture – 55
Microarchitectural Attacks: Part 1 Cach Timing Attacks on Block Ciphers

Welcome to this class on Hardware Security.

(Refer Slide Time: 00:38)

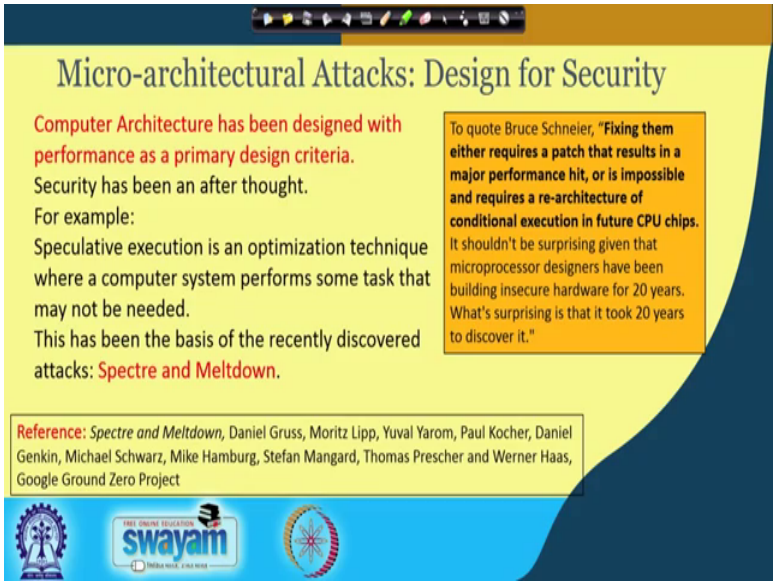
So, today we shall be starting a new topic. We shall be starting to discuss on, what is called as microarchitectural attacks, which basically, is kind of the confluence of architecture or computer architecture and its effect on security of ciphers or cipher implementations.

So, we shall be starting this discussion with quick discussion about what is meant by micro-architectural leakages. We shall be trying to talk about timing channels in cryptography, which is a very important aspect to understand to realize an end to end secured implementation. We shall be discussing in particular about cache attacks and timing channels created using cache memories. And then, we shall be talking about the types of cache attacks and then finally, cache timing attacks on AES.

We intend to also discuss about cache timing attacks on small table ciphers and their impact on of an impact of techniques like parallelization and out of order loading. And

finally, we shall wrap up with a case study on cipher called Clefia. Like a basically a cache timing attack on Clefia which is a very which is again a standard block cipher.

(Refer Slide Time: 01:22)



So, first let us have a quick look on the, you know the effect of micro architectures on security. So, computer architecture has been fundamentally designed with performance as a primary design criteria. And security has always been an afterthought. For example, we have various techniques like speculative execution, execution which is an optimization technique where a computer system performs some tasks that may not be needed. So, this essentially has been the basis of several attacks leading to a very famous attack which has recently been discovered, which is called as which is called as Spectre and Meltdown.

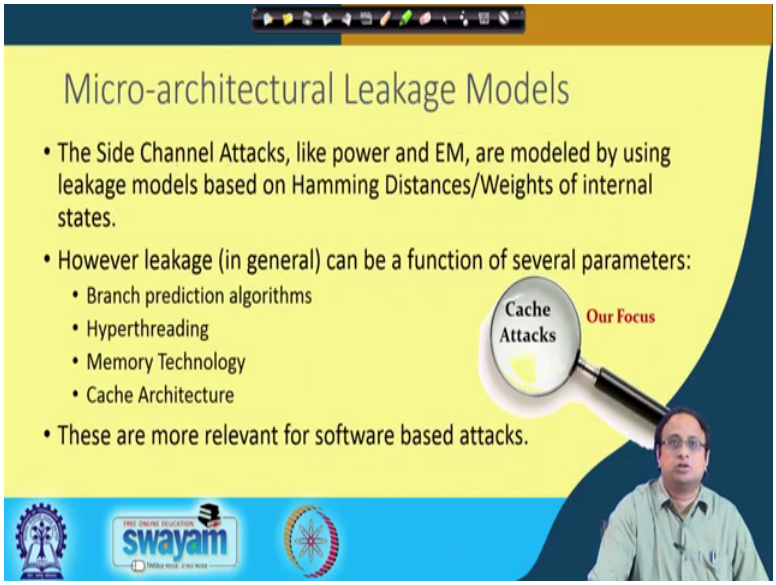
So, so, this is the reference of this attack and this basically kind of shook the world and took the world or most of the, most part of the world by surprise. And to wrap up or to quote Bruce Schneier, who is a very famous security expert and security commenter commentator. Fixing them either requires a patch that results in a major performance hit, or is impossible and requires a re-architecture of conditional execution in future CPU chips.

So, this is essentially extremely difficult because at this point after so many years of evolution of computer architecture and techniques of how to handle such kind of executions or speculative execution. It is kind of you know, like that we have to go back

to 20 years. So, this essentially makes the entire scenario quite dangerous. So, all this happens primarily because you know like the most, even now when we kind of find out or develop a new technique, our primary goal has been performance.

So, security right, essentially cannot be an afterthought and these kind of attacks again and again tell us that we need to have proper design for security criteria.

(Refer Slide Time: 03:08)



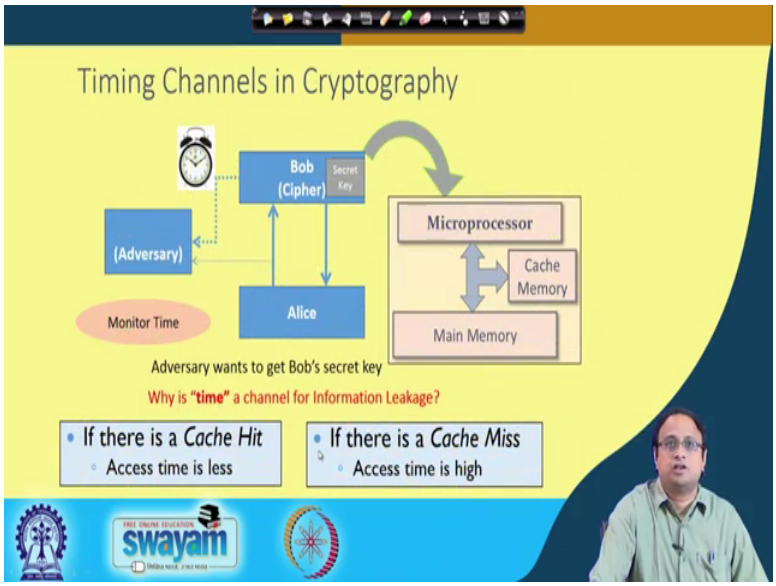
So, so, there are you know like. So, how do you kind of model micro architecture? So, this becomes extremely complicated because of you know, the fact that when we run our processes on computers. Then there are so many complicated things which happen. There are so many complicated interactions that modeling of such kind of attacks become extremely challenging and difficult.

For example, when we are talk previously about side channel attacks where our primary focus was like power, electromagnetic radiations. We found that you know like, we can actually although not exactly accurate, but we can essentially fairly model them by methodologies. Like having distance or having weights of the internal sales. On the other hand leakage right, in when we talk about computers, or you know like computing systems can be actually a function of several parameters. For example, it could be because of the branch prediction algorithms. It could be because of hyper threading. It could be your memory technology. It could be your memory hierarchy. It could be the cache memory or the cache memory architecture and so many. There are so many other

things which can be of direct consequence or direct impact on the, on micro architectural leakages.

So, our focus in today's talk is primarily on cache attacks and in particular about the timing channel which is created due to, due to the existence of cache memories.

(Refer Slide Time: 04:27)



So, therefore, right, if we take a little bit of look and the, and try to understand why cache memory is create a timing channel in pictography. So, let us try to illustrate this by this diagram. So, you can see there are again the two, two members like Alice and Bob, who are essentially participating in a communication. Where you know like, Bob essentially is using a cipher and because of that right, he uses a secret key.

The secret key essentially is embedded in its own computing system and therefore, right the idea is that the adversary should not definitely have access to the secret key. Because, then the enter security collapses. Now, we can pretty much assume that the computing system on which Bob is working, is a general purpose machine which probably from the one humans model essentially look like this.

So, it is essentially you have got a main memory, you have got a micro processor, but we know that there is a memory wall. And therefore, you can pretty much expect, that in order to bridge this memory wall there is an intermediate cache memory which

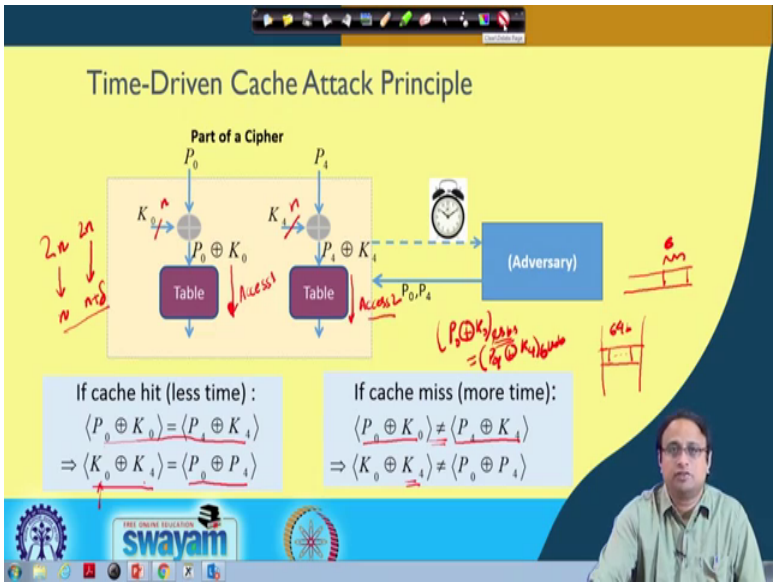
essentially tries to you know like appear as a staircase between the memory and the microprocessor.

So therefore, right? Now imagine that the adversary right, who is basically observing this channel or the communication channel which is untrusted. Has definitely access to the input and the output, but along with it right, imagine that the adversary also has the capability of very accurately monitoring time and measure time or monitor time. So now, the objective of the adversary is again like any attacker or most attackers, is to obtain Bob's secret key. And now, basically the attacker wants to use this time as a channel or a covert channel for understanding what is the internal secret. And therefore, the question is, why is time a channel for information leakage.

So, if you just take a look back or just remember about our classical textbooks in computer architecture we have all seen this, that if there is a cache hit and if there is a cache miss, then there is a difference in timing. For example, if the data is available in the cache memory, then you would expect that the access time of a particular memory access would be less. Whereas right, if there is a cache miss then there are more management in which happens, which results in a larger access time. Because, you access the primary memory or the main memory which is seemingly much slower compared to your cache memory.

So, imagine that the attacker is able to distinguish between a cache hit and a cache miss because of the ability to measure time very accurately. And then, this ability to understand whether an access is essentially resulted in a cache hit or it resulted in a cache miss can apparently reduce the entropy of the secret key.

(Refer Slide Time: 06:59)



So, so in order to again understand that, let us try to take a look about at a specific type of example, which falls into the class of what are called as time-driven cache attacks.

So, again like, observe that suppose, you have a cipher like AES for example, in which there are different parts of the key. For example, K 0 and K 1 here at symbolic or K 0 and K 4 here are symbolic of two parts of the key. Both of them, both of them are getting kind of you know like XOR with you plain text. For example, K 0 is getting XOR with P 0, which is a part of the plain text again and again P 4 is getting XOR with the part which is K 4. And the both of them are creating a virtual address which is say P 0 XOR with K 0 and here in this case it is P 4 XOR with K 4.

So, now imagine that the cipher has been implemented with lot of tables and I will come to this point subsequently little bit more details. But, imagine at this point that the cipher has been implemented using tables. And therefore, right, you make an axis at this table at the location or address which is P 0 XOR with K 0. And here you make an axis at P 4 XOR with K 4. So therefore, right, the adversary is making two independent accesses.

Now, if you kind of remember about for example, the AES structure in which there were 16 parts like when I talk about AES-128. There are 16 parts of the key ok. The idea is that each part of the key is independent. And therefore, right, the entropy is a proportional to 16 multiplied by 8 that is 128 bits. That means, right, that K 0 or K 4 and K 0 and K 4 they to be they should be two independent components of the key. That

means, that if I give you the knowledge of K 0, you basically have no information about K 4 and vice versa.

But, now we will see how using timing channel and you know like just understanding that whether there is a cache hit or a cache miss can significantly reduce the entropy of the key or the key material.

For example, if the, so again you know like the adversary like in most attacks has got a control of the input, we assume that. So, it can basically send pretty much P 0 and P 4 and can also observe the timing by using a very accurate timing channel. So now, suppose, the adversary is able to understand that there is a cache hit by understanding that the suddenly you know like you finds a there is an access time. So, what happens is that there are there are 2 accesses which happens here.

So, the first access for example, happens in this happens in this table right. So, therefore, this is your access 1. So, this is your access 1 and this is the 2nd access which happens here in this table. So, these 2 accesses are seemingly independent. But, imagine right, that the attacker is able to find that access which happens to the 2nd table takes very takes very small amount of time. And therefore, he suspects that it resulted in a cache hit.

So, now like we most of these discussions right, that we do we will assume that the ciphered starts with an initial cache warming, which means that, initially there is no data available in the cache. So, I would expect that so, 1st access definitely resulted in a cache miss and, but the 2nd one can be a cache hit or a cache miss. And suppose, the attacker by its ability to measure time accurately, kind of comprehends that there was a cache hit.

So, that immediately tells us that the address at which the, you know, like the access had happened. Essentially right, if I just assume a very simplistic model of the cache and if I initially assume that there is only 1 element in the cache line. Then, I can directly write that the XOR of P 0 and K 0 is same as that of the XOR of P 4 and K K 4. That means, these two XORs are exactly the same. Even if I assume that, you know, like there are multiple values in the cache line.

Then, as I know that what happens in the virtual address. The position in the cache line is essentially kind of you know, like understood by the lower significant bits of the virtual address. So, that means, right, if there is something which goes into a cache line, then I

would kind of. So, let me make an assumption that I am not able to distinguish between what goes into between the bytes which or between the elements which goes into a cache line.

That means, right, I will be able to recover a certain part of certain part of the address. That means, I can only tell that in that case right, that maybe you know like the, the few the few bits of the of the address essentially resulted in the same value.

That means, what can probably happen in that case right, that when I find a there is a cache hit I can probably tell that the XOR of P 0 and K 0. And if I take the lsb for example, or few lsbs of depending upon how many you know, like. For example, right, if there are like 64 bytes, that it may happen that 6 bits are used to kind of index that. Then, that means, right, the last 6 bits, that means, the six lsbs of P 0 XOR with K 0 is same as that of P 4 XOR with K 4.

So, that means, that so, let me just write 6 lsb. For example, to indicate the least significant bits ok. So, that means, right, it implies that I am not able to recover the entire address, but at the same time like definitely few few bits of the address. So, that basically so so, let us if I if I even if I you know, like forget this complication right now and just assume that there is 1 element in the cache line. Then, I can just write that XOR of P 0 XOR with K 0 is same as that of P 4 XOR with K 4. And therefore, I can write that the XOR of K 0 XOR with a K 4 is same as that of P 0 XOR with P 4.

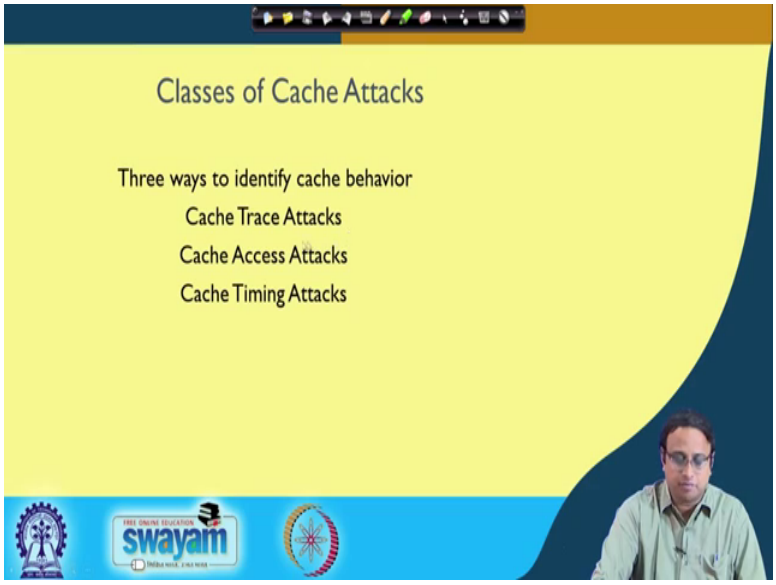
So, what does that tell us? That tells us, that K 0 and K 4 are now not independent. That means, if I have a knowledge of K 0 then I know the value of K 4. So, therefore, right, it implies that essentially what it implies is that now if you assume that the entropy of K 0 was initially n bits and the entropy of K 4 was initially also n bits. That means, initially we are a 2 n bits of entropy.

Now, because of this information leakage, this suddenly shrinks to n. In a more realistic scenario where I consider multiple cache lines ok. So, there so, there what will happen is that suppose, initially you had an entropy of 2 n, that will reduce to n plus delta. So, that delta essentially is because of the fact that, in a cache line there are multiple multiple values which I am not able to discriminate. So, essentially I will not have information about those bits.

But, at the same time right, definitely there is an information leakage because of this. And therefore, that implies that this can be potentially dangerous. Even if there is a cache miss, that means you know, like suppose, an access takes more time I can in that case right, an inequality. Where it means that the XOR of P 0 and K 0 is not equal to the XOR of P 4 and K 4. Even that is a leakage because that tells us that K 4 cannot take certain values which is also equivalent to the leakage of of the unknown key.

So, therefore, right, in both cases we see, even if there is a cache hit or a there is a cache miss there is a potential leakage which we need to kind of take care of.

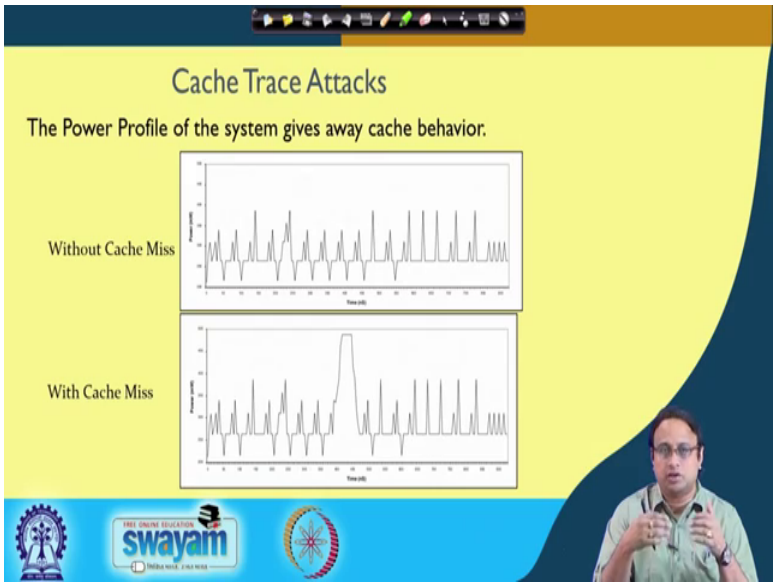
(Refer Slide Time: 14:08)



So, depending upon you know like different kinds of cache attacks or the, you know like different of the presence of cache. There are different types of cache attacks which has been defined. So, here is a quick sort of you know like quick nomenclature of some important class of attacks.

For example, we have got cache trace attacks, we have got cache access attacks and we have got cache timing attacks.

(Refer Slide Time: 25:32)



So, let us take a quick look in between them. So, the cache trace attack essentially stands for the fact that suppose, I do a simulation. And I basically somehow, you know like for all the accesses that a block cipher makes, I am able to kind of understand whether they resulted in a cache hit or a cache miss. For example, right, if I implement AES and in particular, the AES s box as a table. Then, that means right, for example, every round of AES makes 16 look ups or 16 tables accesses.

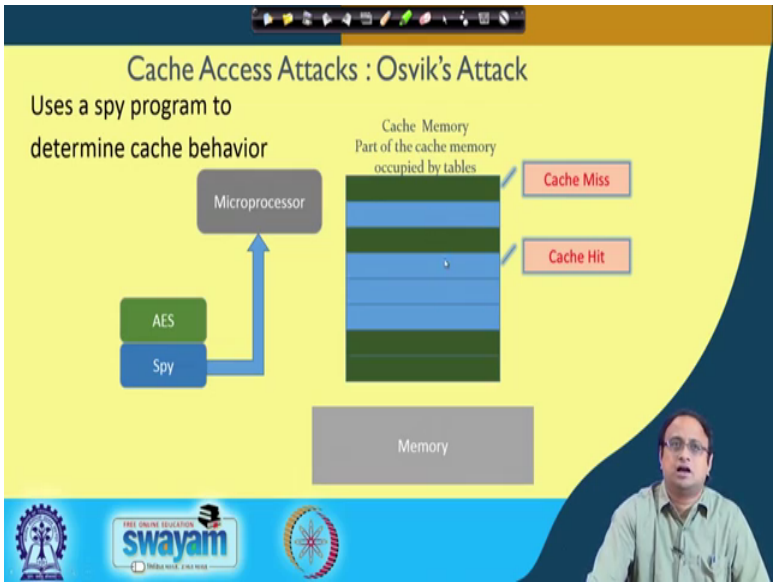
So, if there are like 10 rounds, then I would expect that there are 160 accesses that the AES is making. Or 160 table accesses that AES is making. So, now, now what I am trying to kind of do is, I am trying to kind of do a trace analysis. By you know, like measuring maybe that power consumption of the system. And and I am assuming that if there is an suppose, you know like there is a cache miss. Then, that results in a shoot in the power line. So, what happens is that if I observe the power line and I I see several accesses. Then and suddenly if I get a kind of overshoot, then I reckon that or suspect that, that is because of a cache miss.

So, if I have got this ability, then for all this 160 accesses in AES, I am able to know, you know like the pattern of hits and misses. So, what have been happen is that the 1st access is a miss, then I get a hit, then that is a miss, there is a hit and so on. So, you know, like I have got a trace, I have got a history of you know, like whether the the accesses resulted in a cache hit or a cache miss. So, these class of analysis or these class of attacks,

basically what they do is. They take this cache trace in as an input, then the take the block cipher. And then, they make an analysis to understand whether they they can use this extra information to to retrieve the key ok.

So, these class of attacks are what are called as cache trace attacks.

(Refer Slide Time: 16:23)



So, there is another class of attack which is called as cache access attacks. And probably this little bit more common or well known which essentially resulted from a work by Osvik. And therefore, you can be referred also as Osvik's attack. So, what it does is, basically it uses a spy program to determine the cache behavior. So, it is kind of an attack where we assume that both the target of the victim and the attacker or the spy is essentially co resident and is working on the same hardware platform. It can be pretty much happened particularly right, because of the advent of cloud and other infrastructure which we often share with our adversaries.

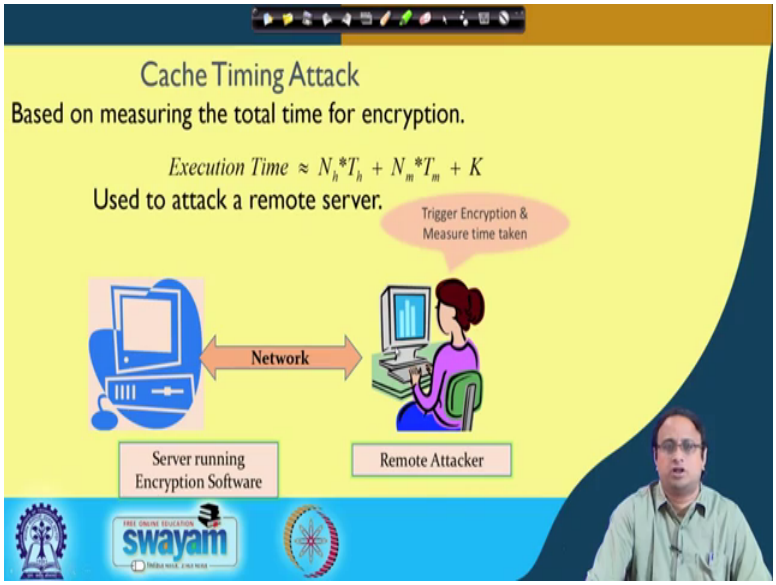
So, in this case, what happens is that, initially the spy makes an access to the micro processor. So, it kind of fills the cache memory with some garbage data and then it allows the AES to execute. So, when the AES executes it of course, makes certain accesses and that implies that it kind of evicts some locations from the cache memory. And therefore, when the spy comes back again and executes. And remember that the since the spy is you know, like when the spy executing and it can again, it remembers the accesses accesses where it originally made. And therefore, it can time its own accesses.

Because, you know like, so the spy is not really timing the you know like, the accesses which AES was making. But, is timing the access accesses which it is it is itself making ok.

So, therefore, right what will happen is that, since these locations where you know, like when the spy comes up and it finds that the time is more. Then, it will potentially suspect that there is a cache miss, which means right the AES has evicted those those locations, those locations from the cache memory. And therefore, it kind of you know, you know we we, will give us the footprint of the AES execution. Where AES executed and where AES did not execute ok. And therefore, right, now we can combine these with something that we already saw. That when we basically build those equations of P 0 XOR with K 0 equaled with maybe P 0 XOR with K 1 or is not equal to P 0 P 1 XOR with K 1. And from there right, can basically reduce the entropy of the entire AES-128.

So, therefore, right this essentially is quite devastating attack. And these kind of attacks are often called as the prime plus probe attacks.

(Refer Slide Time: 18:39)



So, now there are different variants of, as I said different variants of cache attacks. There is another very important class of attacks, which is called as cache timing attacks. And this is even more powerful because it basically kind of claims that it can potentially work over the network. And essentially therefore, can be used to attack a remote server potentially.

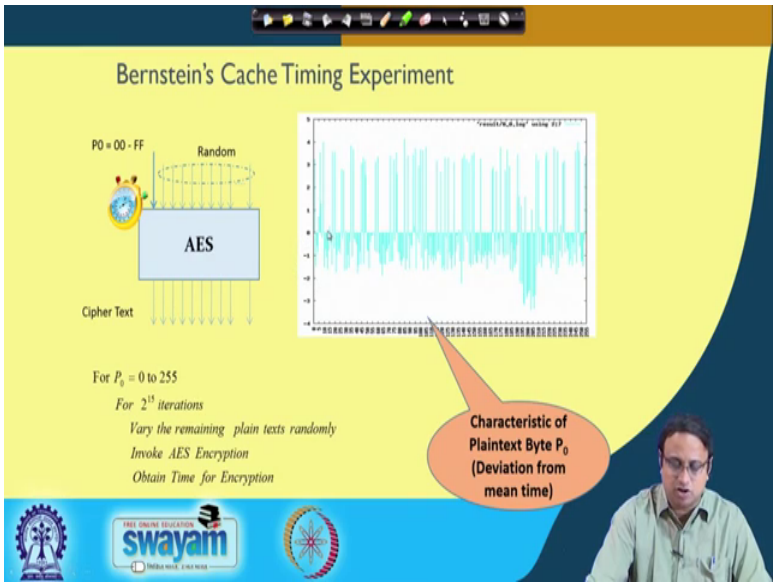
So, this is the again you know, like straight from the text book. We remember that. if I ask you right, that there is a bunch of instructions and then you know like what is the execution time because of that. Then, our text book tells us that if the access time for the cache memory is T h and if the access time for the cache I mean when there is a miss right, is T m. And if the number of hits is N h and the number of miss is T m is N m. Then, the total execution time is is obtained by multiplying N h with T h and then we add N m with T m, plus some noise. This K stands for some thing which I am not able to exactly model. And now imagine that what, this is the scenario of the attack.

So, basically there is a remote server which is running an encryption software. And there is a remote attacker, who basically establishes a network. So, this could be potentially even by something as common as by as as as a TCP network. And then, basically sends packets to the server to encrypt. And then, when the server sends back the cipher text, along with it, it also obtains the timestamp or the time of you know, sending the data and receiving the packet back. That means, you know like, it basically tries to do a statistical analysis or the attacker tries to do a statistical analysis to know what is the time which the encryption took in in the remote server. And then, it basically tries to kind of do a statistical analysis to retrieve the key from there.

So, this as you can see right, is a quite you know like, a very practical attack model because apparently right it can potentially work on over the network. So, as we will see in more details right, in this class of attacks we actually make also an assumption. The assumption is that, the attacker has an access to a similar looking software or similar looking server ok. With exactly the same specifications which it has pre characterized ok.

So, it basically can in in that particular target server, it can therefore, make a characterization with an with an example of a known key ok. And then, it basically targets the victim server where the key is not known. So, it is kind of similar to the template attacks that we have seen in the context of side channel attacks.

(Refer Slide Time: 21:10)



So, this paper or this work was primarily initiated by Bernstein in a work in 2005 which showed that, how we can perform a cache timing experiment.

So, let us try to understand this with a example of AES. So, in AES as we know that, so, in particular AES-128 we know that, there are 16 bytes of input or plain text. So, what we do is, we take one of the bytes for example, say P 0. And I want to obtain the value of K 0. That means, the 1st byte of the secret key. So, therefore, the remaining part, that means, the remaining 15 bytes of the key, I I make the I make several iterations of them ok.

So, therefore, what I do is, I initially fix the value of P 0. And for the remaining 15 bytes I just make large number of variations. So, I probably I I I make 2 power of 15 choices of the remaining remaining plain text bytes. And then, I I I apply it you know, I apply the AES engine on those plain text. And then, I also observe the timing corresponding to each of this encryptions. And then I take the global, I take the I take the average of all these encryption times. And then I plot them in the form of a of of a timing characteristic.

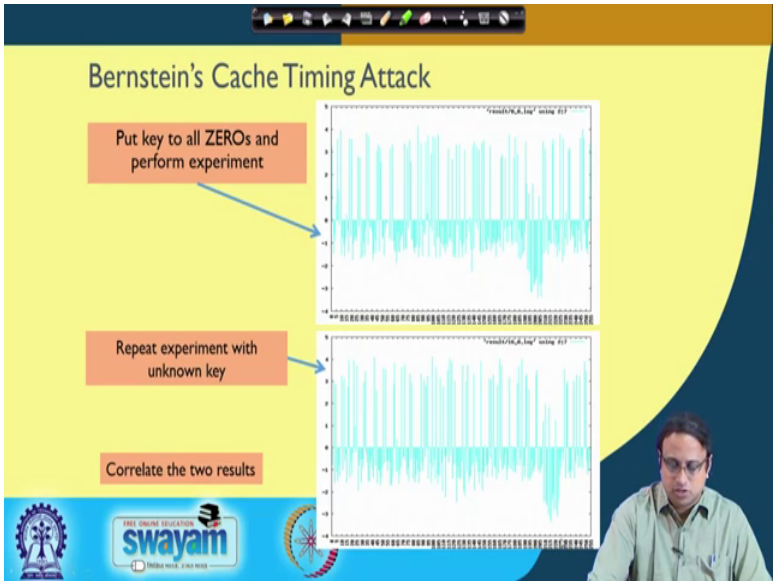
So, what I do is, I basically take all the values of P 0. So, initially suppose P 0 is 0 and then I basically make 2 power of 15 variations of the remaining 15 bytes. I get the average timing. I plot that average time. Likewise, I again change the value of P 0 to something else. And then you know like, again I apply 2 power of 15 randomly chosen values of the inputs, keeping this this value of P 0 as constant to maybe 1. And then I

again obtain an average several timings and then I take the average of that ok. And then I plot that.

So, then what I do is, I I essentially kind of you know like pretty much fix P 0 to all the 256 values. And then, I obtain this graph which is like, which is called as a timing characteristic of the zeroth byte. Note that, when I am observing this timing, in order to remove the effect of noise, what I do is, instead of plotting or you know, like plotting the average time. I also compute the global average and then I deduct the global average from that individual average time.

So, therefore, right, this is just a de-noising technique. And therefore, we obtain the characteristic of the plain text byte b 0 and as written here, we obtain the deviation from the mean time. That means, from the global mean time. So, it is very interesting to see this timing characteristic. For example, you can see that there are certain bundles ok. And potentially these bundles are because you know like, there are some data in the in the cache line which are difficult to distinguish. And therefore, they have a similar kind of timing behavior.

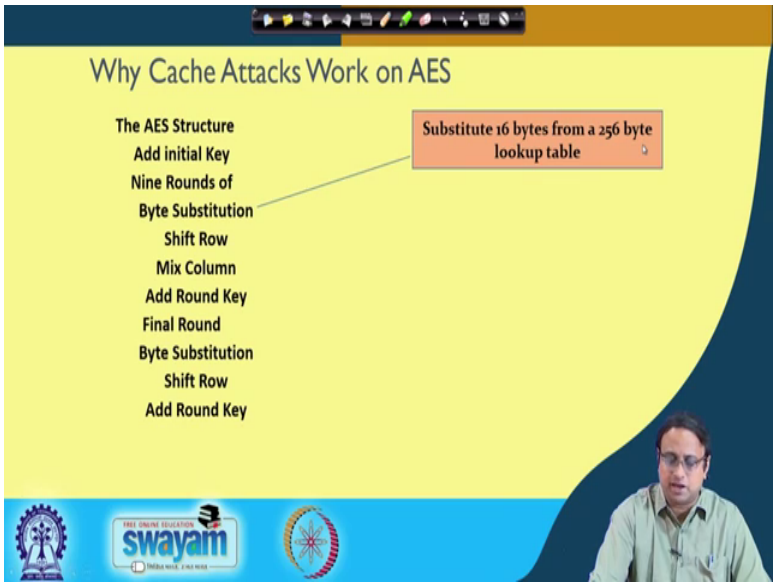
(Refer Slide Time: 23:53)



So, now what what we do is that, again as as you remember that in the cache timing attack model, we assumed that there is a target server. Where you can do a templating process, which means you can fix the key to 0. You can derive a timing characteristic as we have seen. And then now you make a target that means, you do not know the key and

therefore, you repeat this experiment. And then again obtain the characteristic. So, you will find that the characteristics in both cases are pretty simple, except that one of them is a shift of the other. So, if you are able to measure the shift by applying some kind of standard correlation technique right. And if you correlate these two results then potentially, you get the zeroth key byte. Because, you basically get the XOR of this key and this key and since this key is held to 0, you actually get the unknown key ok.

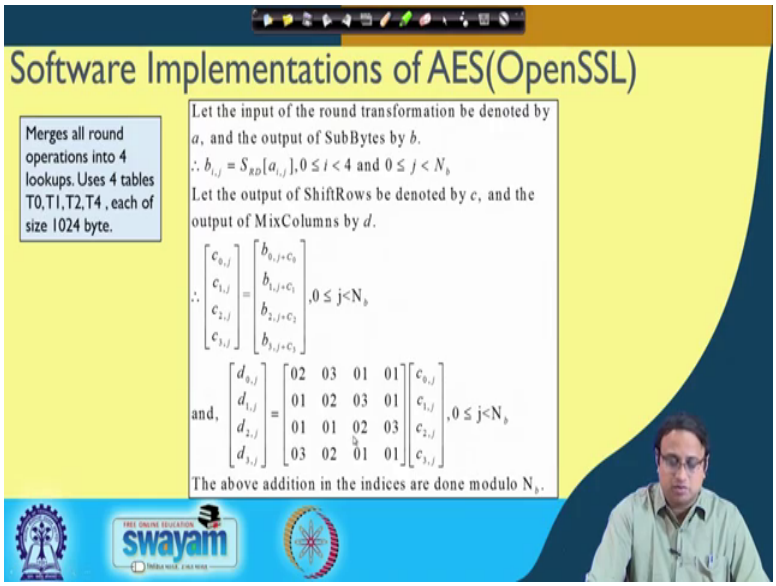
(Refer Slide Time: 24:47)



So, therefore, the shift is what we measured. And the shift essentially stands for your unknown secret key. So so, therefore, this is a very interesting attack and therefore, right, people have kind of tried to analyze why cache attacks work on AES. So, for that right, let us just recollect the structure of AES. This is the structure of AES and then we have to kind of understand how do we implement AES for software.

So, for that right, we note that, this byte substitution is or can be implemented potentially as 16 bytes from a 256 byte lookup table ok.

(Refer Slide Time: 25:15)



And in fact, right, you can actually implement using bigger tables. So, if you just look, take a look into OpenSSL, which is a repository or several cryptographic implementations. And, if you look into the implementation of AES, then you will find that it is implemented or every round is implemented using 4 look up tables.

For example, let the input of the round transformation be denoted by a and the output of the sub bytes by b. Then, we know that every byte like the input byte, is basically getting modified by the S box of Rijndael or AES into say b i, j. So, a i, j here stands for the input and b i, j stands for the output. Again, as you can understand that i can vary from 0 to 4 and j can vary from 0 to 4. So, N b here stands for 4 and therefore, we have got 6 16 cross, 16 byte wise organization of AES as we have already previously seen.

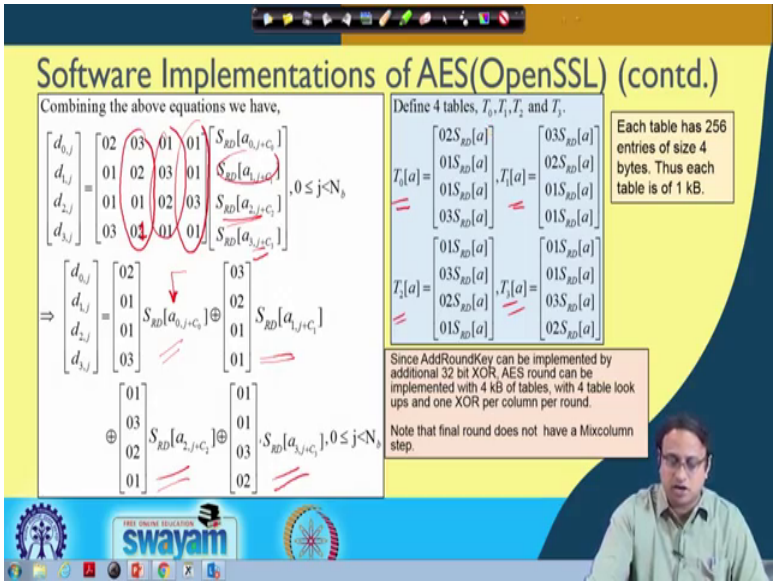
So, now if we apply the shift rows, then we know that every byte here gets shifted. For example, this is essentially b 0, j plus c 0. So, c 0 for the zeroth row stands for 0 because in the zeroth row there is no shift.

For the 2nd row there is a shift by one location. So, c 1 stands for 1, c 2 stands for 2 and c 3 stands for 3. So, therefore, right, this column stands for b 0 comma j plus 0, that is j. This stands for b 1, j plus 1. This stands for b 2, j plus 2. And this stands for b 3, j plus 3. And therefore, right, if we, in that way you basically accommodate the shift rows operation. And then you have to perform a mix columns. The mix column basically performs on this particular column. And you basically multiply this matrix, which is

already pre known. You multiply with this particular column and you get the values of ds ok.

So, this is essentially your output. So, now, the question is right, you can definitely write a code where I do this operations individually. But, it turns out that it in software it becomes much more efficient if we can implement the entire mapping by look ups.

(Refer Slide Time: 27:17)



So, therefore, right, here what we do is, is illustrated over in this particular slide. So, you can see that, what we need to do is this. Because, this essentially is nothing, but S RD applied on the corresponding input bits. So, S RD stands for the S box of AES or S box of Rijndael.

So, therefore, right, any byte here like, this column for example, can be obtained by multiplying this column like 2, 1, 1, 1, 3 with this byte ok. So, this is the byte standing for the zeroth value or the zeroth row. And likewise, the 2nd, and then you XOR this with this column, that is, 3, 2, 1, 2 multiplied by this should be 3, 2, 1, 1, 1 actually ok. Because, I I should multiply 3, 2, 1, 1 here. So, let me correct this. This is 3, 2, 1, 1 ok, 3, 2, 1, 1.

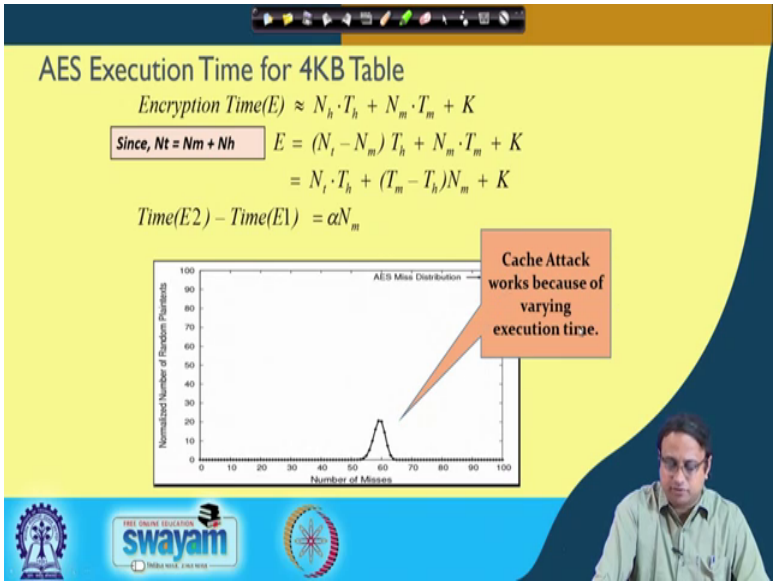
So, we basically multiply this column with this byte and that is essentially shown over here ok. Likewise, you multiply this column with this byte and that is shown in over here. And finally, you multiply this column with this byte and that is shown here.

So, note that each of these look ups essentially takes a 256 bit, I mean an 8 bit input. And results in a 32 bit output ok. And these mappings can be shown here as 4 tables, T 0, T 1, T 2 and T 3. And this essentially stands for look ups where the input is 8 bits and the output is 32 bits. So, therefore, it turns out that every table here takes 1 kilobyte amount of memory. And therefore, totally right, it takes 4 kilobytes amount of memory.

So, therefore, the AES in OpenSSL essentially has been implemented using these tables. And that implies that, there are 4 table accesses that you are doing per round ok. And that implies that, if you take 9 rounds, remember the last round cannot be implemented in this way because in the last round there is no mix columns. So, therefore, right, you are making 4 into 9 4 into 9 that is 36 accesses of various look ups ok

So, there is a significant amount of cache activity that happens because of these kind of implementations which can be targeted by the attacks that we are currently discussing.

(Refer Slide Time: 29:37)



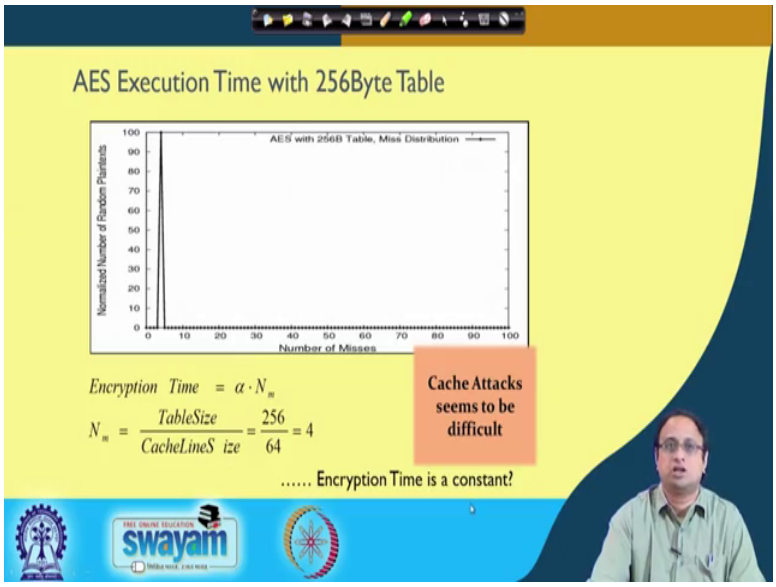
So, therefore, right I mean so, if you do this then potentially what we what. So, if you now take therefore, you know like try to execute AES with this 4 kilobyte table implementation and consider two different runs of the encryption. So, remember if I just kind of store that as N t as a total number of you know, accesses. Then, we know that N t is nothing, but the summation of the number of misses plus the number of hits in an access.

So, therefore, N t is a constant right, because the total number of accesses that you do to the tables remains a constant for that algorithm. So, therefore, what I can do is, I can replace N h with N t minus N m. And therefore, I can rewrite this equation as N t into T h plus T m minus T h into N m plus that constant K. So, if I take two different runs of the encryption and if I kind of take the difference between these two runs, then this part would cancel because, this part is a constant. I will only have an effect on or proportionality the on the number of misses.

And what we find is that, if you take an if you do these experiments. That means, if you take the number of misses and do a frequency plot. You will find that there is a nice distribution of the number of misses which means there are number of misses varies across execution. And that is partly happening because you know like, your table is big. And therefore, right, it may happen that you know like, the entire table right, essentially is creating this nice distribution. And therefore, because of this various variation in the number of misses, you actually get varying times, which can be exploited by the adversaries to know whether there is a cache hit or a cache miss. And also to perform you know, like a correlation kind of attack that we see in the case of Bernstein's timing attack.

On the other hand right, it is very interesting to know. So, cache attacks right, we kind of suspect will work because of this kind of varying execution time.

(Refer Slide Time: 31:18)



What if I implement AES with a very small table, like maybe a 256 byte. That means, I just implement the S boxes table and the other things I do by computations. If you repeat this experiment, then you can expect that the now, the number of misses will become a constant. Because, this table size is not 256 and if you expect for modern cache has right, you will have a cache line size something like 64. Then there will be like, 4 number of misses that can happen. And therefore, right, as we know that there are several accesses that you are doing, like 36 accesses. That means, like 4 will remain a constant, that means, every time you do this encryption, you will always get 4 misses ok. And therefore, the variation will go.

So, for example, if you make a plot now, you will find that around 4 you will get a peak and there will be no other things that will occur. So, the question is right, can you still do an attack on this? Because, the encryption time looks like constant. And therefore, if you vary, then it seems that the cache attack becomes much more difficult. So, now, what we will study in the next discussion is that, how we can still attack these kind of implementations. In particularly, what we what we shall be discussing is that, we shall be trying to look into some modern micro-architectural techniques, which actually helps us in doing attacks even in those cases. So, that we will take up in the next class and.

Thanks for your attention.