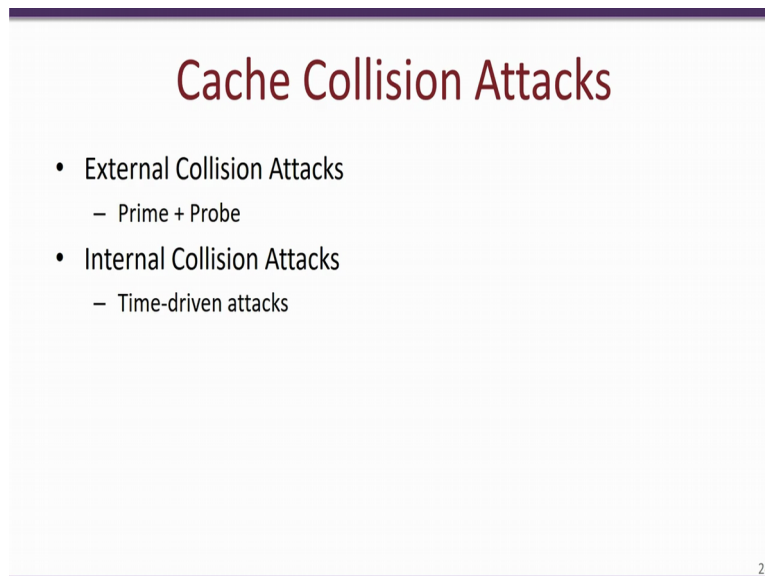


Information Security - 5 - Secure Systems Engineering
Professor Chester Rebeiro
Indian Institute of Technology, Madras
Prime_probe attacks

Hello and welcome to this lecture in the course for secure systems engineering. In the previous lectures we have been looking at cache timing attacks, we had looked at the cache cover channel first and then (we) in the later video we had looked at the flush plus reload attack. In this lecture will be looking at another cache timing attack known as cache collision attacks.

(Refer Slide Time: 0:45)



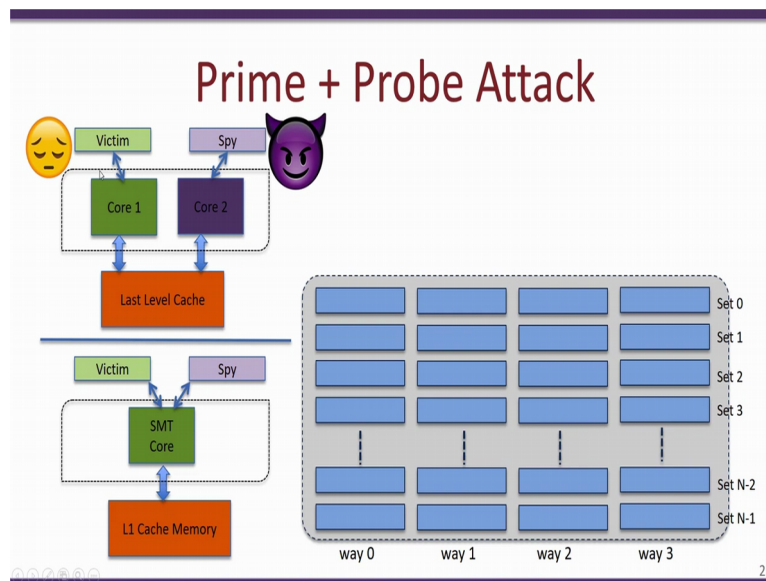
Cache Collision Attacks

- External Collision Attacks
 - Prime + Probe
- Internal Collision Attacks
 - Time-driven attacks

24

So essentially there are two types of cache collision attacks, they are external cache collision attacks and internal cache collision attacks. So external cache collision attacks are popularly known as prime and probe attacks, while internal collision attacks are known as time-driven attacks, so in this lecture we will first start with a prime and probe attack and then we will look at time-driven attack.

(Refer Slide Time: 1:12)



In a prime and probe attack the scenario we are considering is either this or this or we have a victim process running in a particular core and a spy process running in another processor core, the both the victim and spy are sharing the same cache memory. So in this particular example we have the last level cache memory shared between the victim process and the spy process.

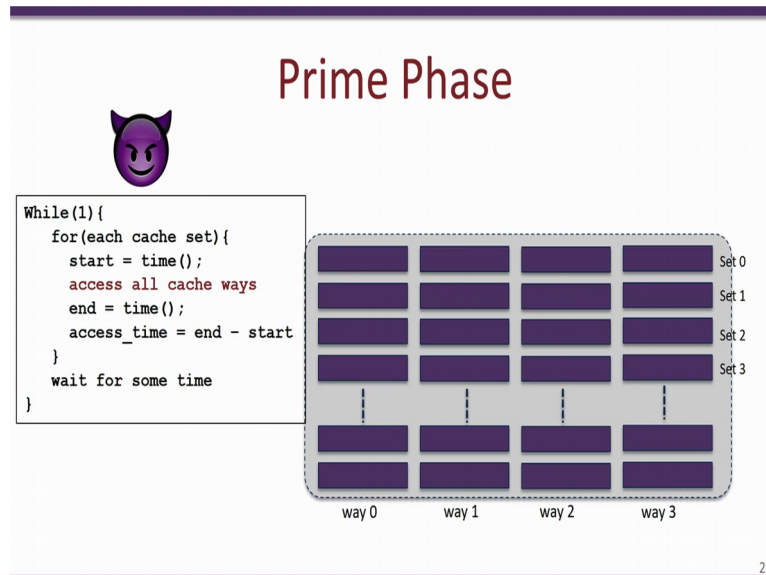
In this setup on the other hand we have both the victim and the spy running on the same processor core, the assumption over here is that this particular processor core is a symmetrically multi-threaded processor core or in other words when using the Intel's nomenclature this processor core is a hyper threaded processor core. So given this particular scenario we have the victim and the spy running simultaneously and sharing the common L1 cache memory.

As we have seen before the cache memories could be very abstractly represented by this particular figure. So over here we have rows corresponding to each set in that particular cache memory, so here for example we have N cache sets going from set 0 to set N and each set has 4 ways, way 0 to way 3, so depending on the address that is accessed by the victim or the spy process so one particular cache line in a particular set is used to store the victim and the spy data.

Now what we will see in this lecture is that in a prime and probe attack in situation such as this it is possible for the spy process to gain some information about the victim process. So

this attacks has been used very famously retrieve a (())(3:26) from a victim process which is executing a cryptographic algorithm.

(Refer Slide Time: 3:32)




The prime phase looks something like this, so if there is an infinite loop in the spy process and for each cache set the spy process creates certain load instructions, which accesses all the ways of that particular cache set, while this is done the spy also times these load instructions. So the timing is done by this function, where initially we invoke a clock source and get the starting time and then after accessing all the cache ways and loading all the spy data into the cache then we invoke the timer again and get the end time, now the access time is given by end minus start.

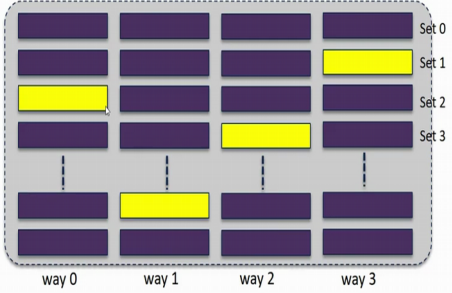
So once this is done for all the cache sets what good result at the end of this for loop is that the entire cache is filled with spy data, so spy is continuously doing this and as a result the entire cache is filled with the spy data. Next what happens is that the spy process waits for some time, the next what happens is that the spy process waits for some time and is essentially waiting for the victim process to begin execution.

(Refer Slide Time: 4:50)

Victim Execution



The execution causes some of the spy data to get evicted




27

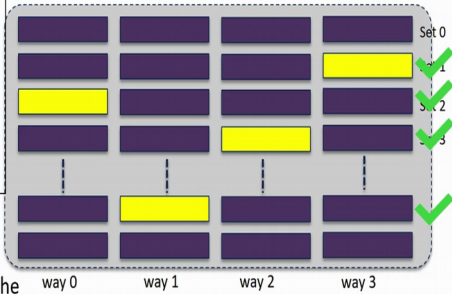
Now the victim process during its execution would start to access certain instructions. Now when the victim executes there will be certain sets in the cache memory, where the spy data would be evicted and replaced with the victim data, cache memory would typically implement some replacement policy such as the least recently used and identify a particular cache line to evict and that particular cache line is replaced with the victim data.

(Refer Slide Time: 5:28)

Probe Phase



```
While(1){  
  for(each cache set){  
    start = time();  
    access all cache ways  
    end = time();  
    access_time = end - start  
  }  
  wait for some time  
}
```



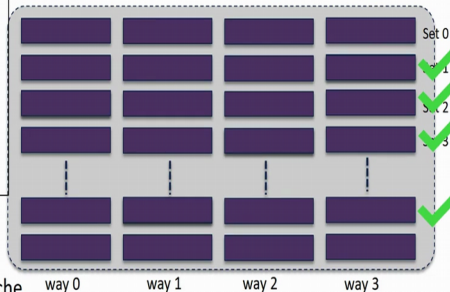
Time taken by sets that have victim data is more due to the cache misses

28

Probe Phase



```
While(1){
  for(each cache set){
    start = time();
    access all cache ways
    end = time();
    access_time = end - start
  }
  wait for some time
}
```



Time taken by sets that have
victim data is more due to the cache
misses

28

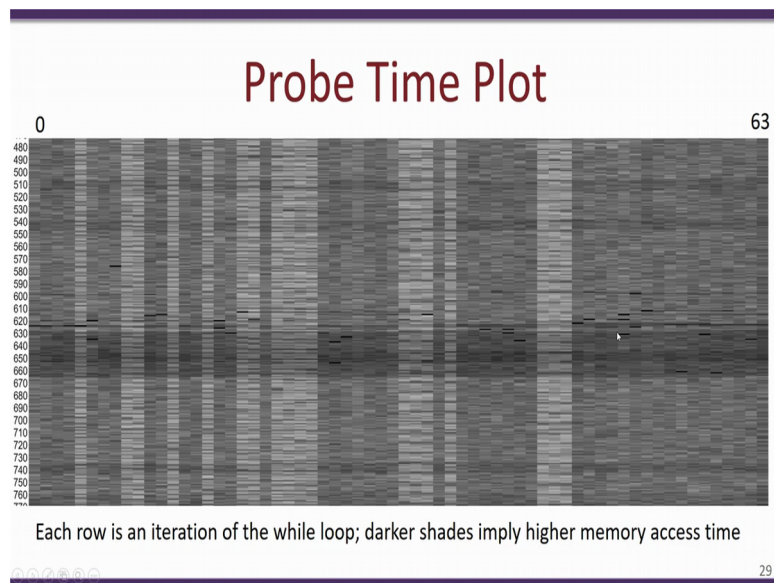
Let us consider what would happen when the victim execute again. So after waiting for some time the victim enters the probe phase. So in the probe phase which is again this for loop being executed the victim would start to access every cache set and in the probe phase the victim would parse through every cache set and access all the cache lines present in that particular set and at that time it would also measure the time required to make these accesses.

Now if we actually look at this and try to infer what the spy sees, if we would see that when the spy accesses this set that is a set 0 the time taken would be less because the contents of set 0 corresponds to all spy data and therefore the spy would only incur cache hits and as a result the access time for set 0 would be low.

Now the access time for set 1, 2, 3 and N minus 2 would be high, the result is that the spy would incur certain cache misses and the cache misses are incurred because the spy data has been evicted and replaced with the victim data and during the probe phase there would be further cache misses incurred by the spy process in these sets specifically sets 1, 2, 3 and N minus 2 in this way the spy process would be able to identify this cache sets which have victim data or in other words the spy process would be able to identify the cache sets which the victim has accessed.

As a result of the probe phase victim data which was present in the cache gets evicted out and replaced again with the spy process. And this prime and probe face gets continues over and over again as we see over here.

(Refer Slide Time: 7:34)



So this particular plot shows the time obtained for the probe phase, so each row over here corresponds to one particular iteration in the spy process and each column corresponds to a particular set.

So for example this was run on an Intel i7 machine which had an L1 cache with 64 cache sets, so each column over here corresponds to a particular cache set. Now a lighter color would indicate that the execution time measured was low. In other words a lighter color would indicate that the spy process in that particular iteration had observed cache hits. On the other hand a darker color such as these over here would indicate that the execution time was higher in which case the Pi process has incurred cache misses.


So we can actually clearly see that there are some cache sets where clearly cache misses are always observed. For example looking at this particular plot we can actually infer the sets which had incurred a high execution time. For example all the darker cache sets like these over here have a higher execution time indicating that the spy process has incurred a lot of cache misses. On the other hand the lighter shades like this cache set and so on have incurred a lot of cache hits and therefore are a lighter shade.

In this particular way by tracing the how the cache memories have been accessed would get an indication of what the other process is doing. So another example is these regions at this particular point, which are considerably darker. So this regions would mean that there is some activity for example another process that was getting context which or something like that

which has been executing and performing a lot of memory operations and thus we see that it has affected the spy process.

(Refer Slide Time: 9:50)

Prime + Probe in Cryptography



```
char Lookup[] = {x, x, x, . . . x};

char RecvDecrypt(socket){
  char key = 0x12;
  char pt, ct;

  read(socket, &ct, 1);
  pt = Lookup[key ^ ct];
  return pt;
}
```

Key dependent memory accesses

ct: 0xFF
key: 0 → lookup [0xFF]
key: 0xAA → lookup [0x55]

The attacker know the address of Lookup and the ciphertext (ct)
The memory accessed in Lookup depends on the value of key
Given the set number, one can identify bits of key ^ ct.

30

So now we would see how we could use this prime and probe scheme to retrieve the secret key in a particular decryption algorithm. Let us say we have a function which looks something like this, so the function is called RecvDecrypt and it takes a particular socket and over here we define a secret key which has a value of 12. Now what we do is we read the ciphertext from that socket and this ciphertext is stored is just a character which is stored in ct.

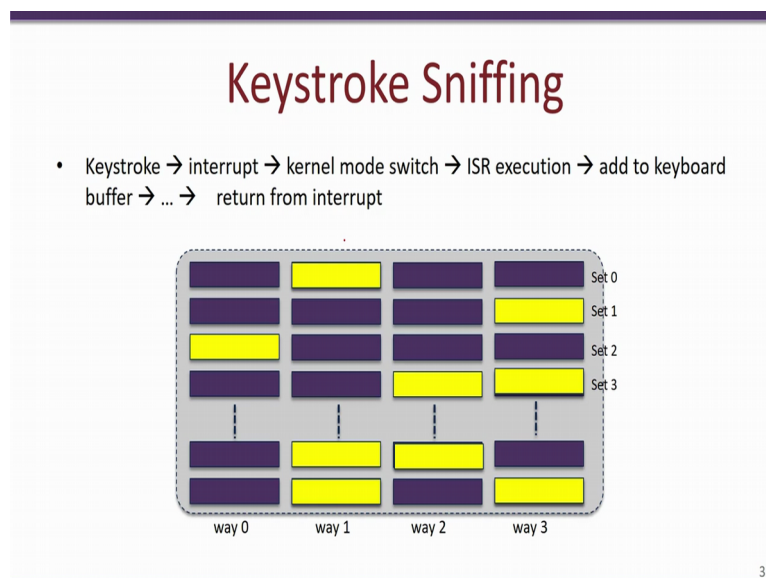
Now there is a lookup on this particular array with containing some character values which we have not specified over here and the lookup is based on an index which is key X or ciphertext, this operation is used to obtain the plaintext which is then returned. Now this is just a toy decryption algorithm, which you have just built up to show how prime and probe can be used to break cryptographic schemes, the important point for us to observe over here is that this lookup to this particular array is on a address location which is key dependent.

So for example assuming that the attacker knows the value of the ciphertext, index of this particular lookup would depend on the value of the key. So for example if let us say the ciphertext has a value say 0 x FF if the key value was 0 lookup is to the location 0 X FF. On the other hand if the key had the value 0 X AA then the lookup is to a location 0 X 55, thus you see that this lookup operation over here is essentially a memory access, right the this memory access is going to load a different data in the cache memory.

Now assume that we are actually running this particular process and this is our victim process and side by side we also run a spy process which is continuously running the prime and probe scanning the measuring the time taken to make memory accesses to a different sets in the cache. Now if the key is 0 X AA and the attacker knows that the ciphertext is 0 X FF, then corresponding to this lookup plus 0 X 55 the spy process would see an increased number of cache misses.

So this would be observed by an increase in the execution time for that iteration in the spy process and thus the attacker can infer that the victim process has accessed that portion of memory and from that could infer that the key value is either 0 X AA or something very close to 0 X AA.

(Refer Slide Time: 13:25)

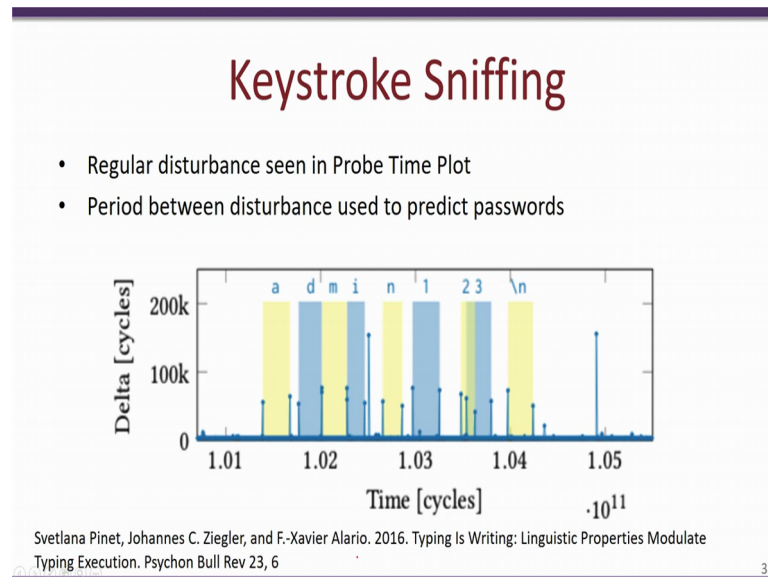


Besides cryptography the prime and probe type of attack have been used for a larger number of different applications, one other famous application is for keyboard sniffing, so this particular process is the victim process. Now you have the other process which is our the attack process which is doing the prime and probe operations. So whenever there is a keystroke that is whenever a user presses a key for example let us say he is pressing a key with respect to his password, each keystroke results in an interrupt the interrupt results in a switch to kernel mode, there is an ISR that gets executed and so on. So each keystroke results in a lot of execution that takes place.

So whenever a keystroke is pressed it results in a lot of different functions both in the operating system and both in the victim application that gets executed, as a result we would

have a lot of the spy data which is running the prime and probe to be evicted from the cache thus the attacker would be able to identify the instant at which the keys on the keyboard were pressed.

(Refer Slide Time: 14:29)



This is a plot from a recent paper which is shown here and what we see over here is the delay in clock cycles and over time. So we see that as keys are being pressed what the prime and probe application sees is that there is an increase in execution time during a particular place. So for example when no keys are pressed the execution time which is measured by the spy process is low and when a particular key is pressed then we see an increase in the execution time as you see in these instances.

Again after some time when there is no key pressed the spy does not see an increase in the execution time, there are of course a lot of errors which may also creep up like for example this over here which shows an increase in execution time even though no keys have been pressed.

(Refer Slide Time: 15:24)

Web Browser Attacks

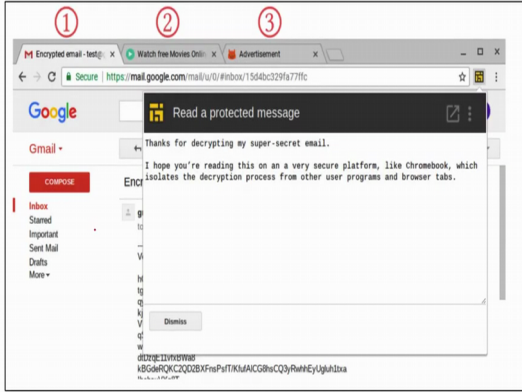
- Prime+Probe in
 - Javascript
 - pNACL
 - Web assembly

33

So the prime and probe attack has also been used in various other use cases especially it has been used to create to attack JavaScript, native client and web assembly on web browsers such as the Google Chrome and Firefox.

(Refer Slide Time: 15:40)

Extract Gmail secret key



<https://www.cs.tau.ac.il/~tromer/drivebycache/drivebycache.pdf>

34

So what this particular slide is taken from this paper over here which is known as the drive by cache and it actually showed how the Gmail secret key can be retrieved by using a prime and probe attack. So what would typically happen is that the user is made to go to a particular link and click on a particular advertising website and this website would open a tab which is an advertisement tab and maybe show display an advertisement but also in the background it would be running the prime and probe attack.

So whenever there is an encryption or decryption that takes place the prime and probe would measure the activities on the cache memory and use that timing to infer secret information.

(Refer Slide Time: 16:31)

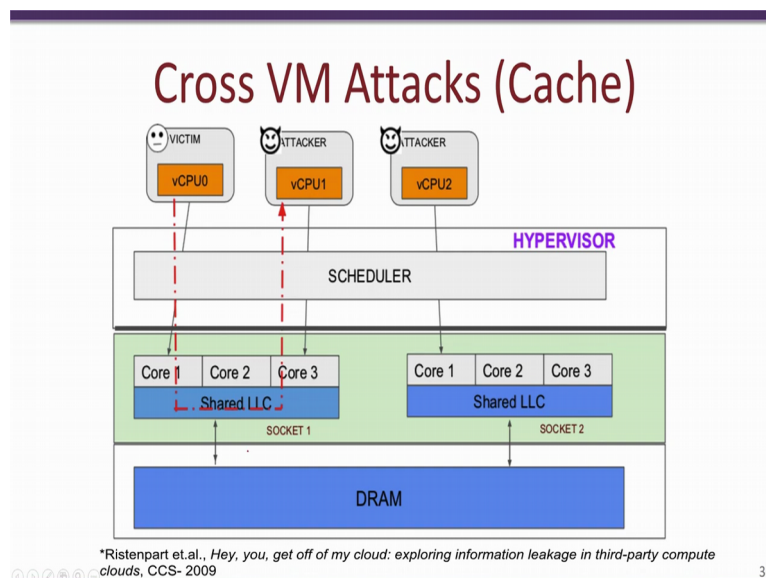
Website Fingerprinting

- Privacy: Find out what websites are being browsed.

35

Another famous application of the prime and probe attack was is for website fingerprinting where the prime and probe application can identify what websites are being browsed.

(Refer Slide Time: 16:45)

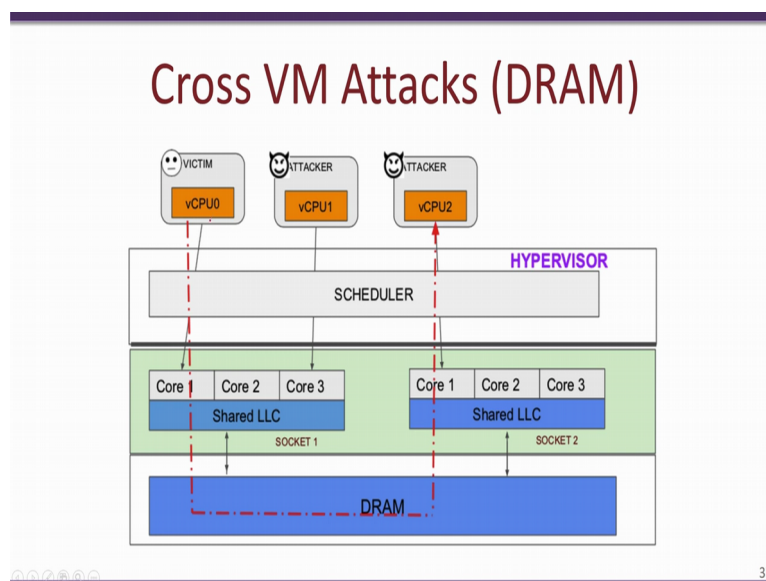


Another very famous attack which was first presented in this particular paper by Ristenpart in 2009 determines how cross virtual machine attacks can be done using something like the prime and probe attack, here we are using a cloud environment which have different virtual machines but the underlying hardware is the same. For example in this figure over here we

would have an attacker sharing the same cache memory as a victim process. Thus the attacker runs a prime and probe application and is able to monitor the cache and memory activity by the victim process.

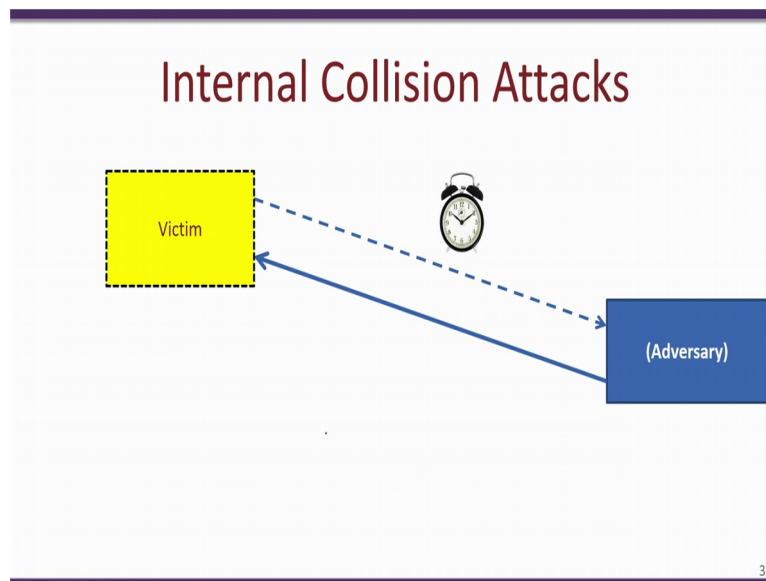
And from this it would be able to identify secret information like cryptographic keys, it would identify what characters have been pressed, or it would be able to sniff keystrokes and so on.

(Refer Slide Time: 17:40)



Very recently a very similar type of attack was also showed using a shared DRAM in such an case the victim and the attacker do not have to share the same LLC but have to share a common DRAM.

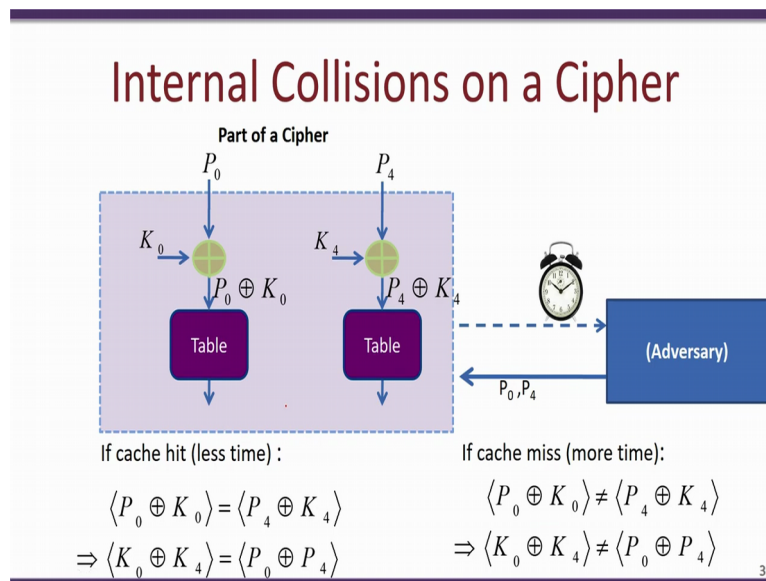
(Refer Slide Time: 17:56)



So now we look at internal collision attacks or properly known as time driven attacks, here the scenario looks very different, here what happens is that the victim and the attacker may not necessarily share the same computing resource, or may not necessarily share the same cache memory. So we will now look at internal collision attacks these are properly known as time driven attacks.

So in such a scenario we have an attacker who is able to invoke a victim application and is able to measure the time taken for the victim to provide a response. So in other words the attacker would send a request to the victim process which may be in a completely different machine and then it measures the time taken for the response to be obtained the differences in execution time that is measured at the attacker's end is then used to obtain some sensitive information about the victim. So these time driven attacks are especially popular for breaking cryptographic algorithms and we will take one very simple example of this scheme.

(Refer Slide Time: 19:16)



So let us say that we have a small part of the cipher which looks something like this, we have a plane text P 0 and a plane text P 4 which is the input to the cipher. Now these inputs each are of let us say a bite wide gets (())(19:33) with keys K 0 and K 4 respectively, thus we get P 0 XOR K 0 at one side and P 4 XOR K 4 on the other side. Now this P 0 XOR K 0 and P 4 XOR K 4 is then used to index into a table. So there is a lookup in this table based on this particular index.

Now let us see what happens when we execute this particular program. So first of all we have the attacker over here and we will assume that the attacker is able to choose the values of P 0 and P 4 as required, so the attacker chooses P 0 P 4 for and triggers an encryption to occur. So during the encryption these operations take place and then the attacker measures the time for this entire encryption.

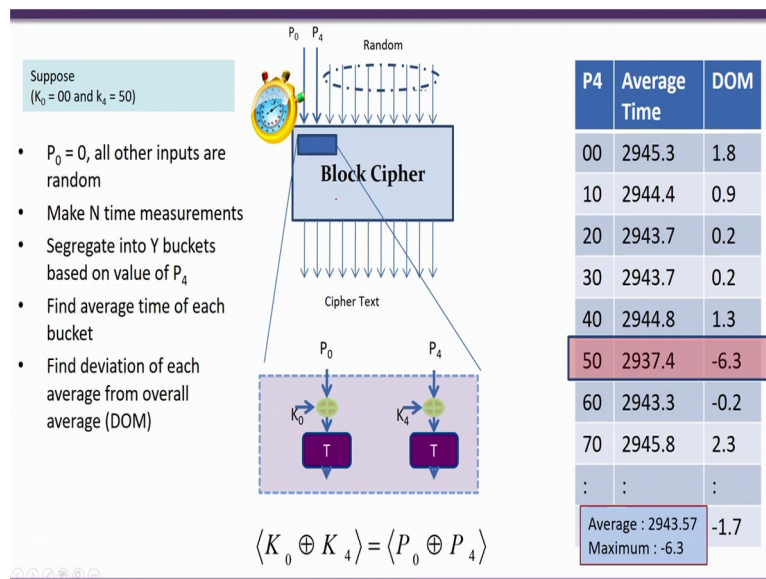
Now we will see how this execution time can vary depending on the values of K 0 and K 4. So first let us assume that the cache is clean and no part of the cache comprises contains any of this table information. So during the first axis at the location P 0 XOR K 0 that would result in a cache miss and one block of memory would get loaded into the cache, the second axis at the index P 4 XOR K 4 could either result in a cache hit or a cache miss, a cache hit is incurred when the index P 4 XOR K 4 exactly collides or is very close to this value of P 0 XOR K 0, in such a case it would indicate that this second axis is to the same or to a neighbouring location as the first axis.

Since this data is already present in the cache, therefore we obtain a cache hit and the execution time for this second axis would be considerably smaller. On the other hand if $P_0 \text{ XOR } K_0$ is not equal to $P_4 \text{ XOR } K_4$ then the second axis would also result in a cache miss second axis would also require a memory block to be loaded into that particular cache, to evaluate the entire execution time we see that we either get one cache miss and one cache hit or two cache misses. And thus we see depending on the value of $P_0 \text{ XOR } K_0$ and $P_4 \text{ XOR } K_4$ the execution time of this particular cipher would vary.

So the based on this variation we can obtain some information about these secret fields, specifically if we have a cache hit then we obtain this relation that $P_0 \text{ XOR } K_0$ is equal to $P_4 \text{ XOR } K_4$ and if we just rearrange the terms we get $K_0 \text{ XOR } K_4$ is equal to $P_0 \text{ XOR } P_4$. Now since the attacker knows $P_0 \text{ XOR } P_4$ he thus knows the XOR of $K_0 \text{ XOR } K_4$. So how does this help the attacker? Suppose we assume that each key K_0 and K_4 is of 8 bits, therefore before running or without running this particular attack the uncertainty of the attacker is 8 plus 8 that is 16 bits.

Now after running the attacker if the attacker identifies this cache hit and obtains a relation like this uncertainty reduces from 16 bits to 8 bits. Now all that is required is the attacker identifies any one of them that is let us say he identifies K_0 and using that K_0 he can easily identify what K_4 is using this relationship. Similarly if there is a cache miss then a relation such as this is obtained and it would mean that $P_0 \text{ XOR } K_0$ is not equal to $P_4 \text{ XOR } K_4$ which means that the index accessed by this in this axis and this axis are not the same and therefore $K_0 \text{ XOR } K_4$ is not equal to $P_0 \text{ XOR } P_4$. So this was a toy example and such an example could be extended to a complete cipher.

(Refer Slide Time: 24:10)



So in a complete cipher such as an AES cipher a very small part of this entire block cipher is having a structure as we have seen before. So for example if you consider a AES and AES comprises of 16 bytes of input and it would give you 16 bytes of output and there are several operations which are actually going on inside the AES block cipher out of which the operations that we were actually interested in looks something like this and is a very small component of the entire cipher.

So the way we actually extend the attack that we seem to a complete block cipher is as follows. So we keep P_0 a constant and keep changing the values of P_4 , so in this particular case we have let us say taken the K_0 to be 0, K_4 to be 50 let us assume that this is our secret information. Now we keep the value of P_0 as constant and for each different value of P_4 we change the remaining inputs randomly, so let us say we change for over like 2^{16} different values of this data for a specific value of P_4 , we measure the execution time required for the cipher like AES to encrypt that particular plaintext using its secret key.

After we do this we find the average time for each value of P_4 . So for example we have 16 different values of P_4 starting from 00, 10, 20, 30 all of these are hexadecimal values and it would end up in F0 for each value of P_4 we would compute the average time that is obtained when all of the other input values are varied randomly for over like say a large number of times say 2^{16} or so. What we notice is that when the value of P_4 becomes 50 we obtain the highest deviation from the mean, so the mean over here is 2943.57 and the maximum deviation from this particular mean is obtained when the value of P_4 is 50 and in

this case we consider the absolute value of the difference of mean which is and therefore it should be 6.3.

So from this what we can see is that P 0 we have set to 0, P 4 we have obtained 250, so P 0 XOR P 4 is equal to 50. So thus we can infer that K 0 XOR K 4 would be equal to 50 and if we go back to what we have kept the values of these keys we have K 0 and K 4 is 50 and thus we see it matches the results that we obtain. So in this way we had looked at cache collision attacks, we had looked at the collision attacks which are external which requires a spy process to execute in the same system as the victim process and also share the same cache memory as the victim, we also looked at internal collision attacks where an attacker need not run a spy process it only trigger the victim program to execute and measure the amount of time it took for that particular victim to execute and based on this time the attacker would be able to infer secret information about that victim process, thank you.