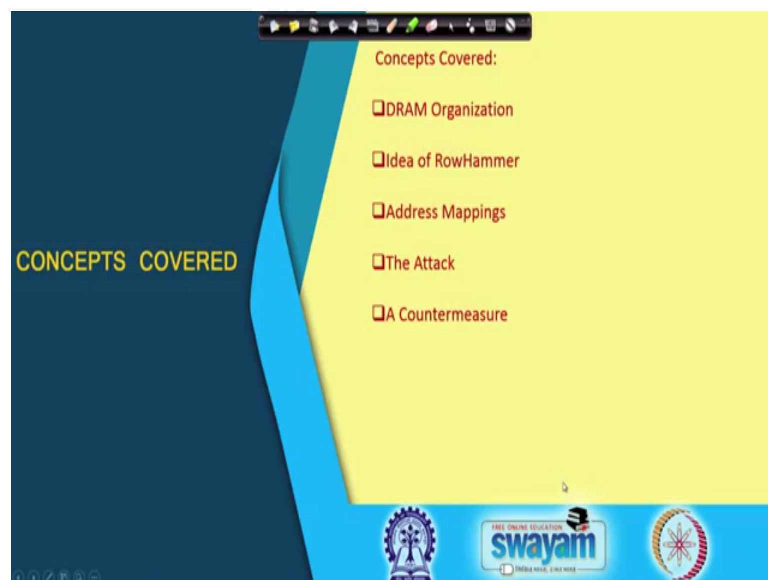
Hardware Security
Prof. Debdeep Mukhopadhyay
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 59
Microarchitectural Attacks: Part 3.
Row Hammer Attacks

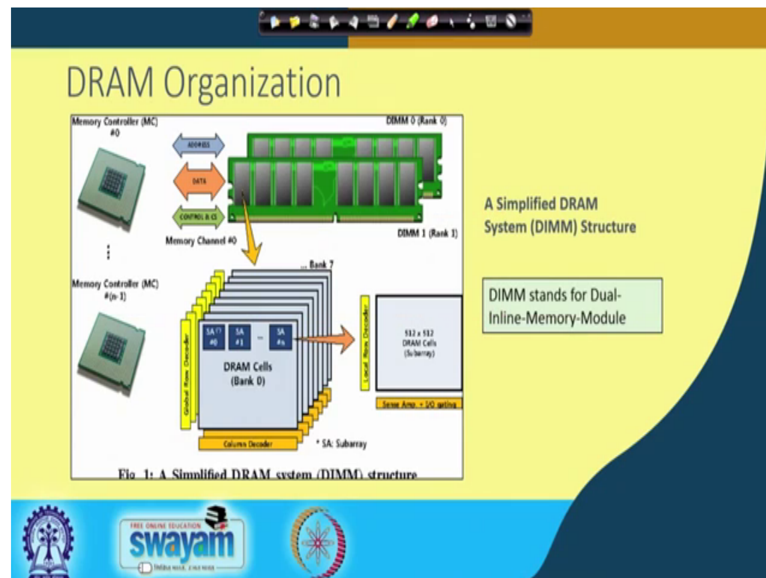
So, welcome back to this class on Hardware Security. So, we shall be continuing our discussions on Micro architectural Attacks and we shall be today studying an attack which is called as Row Hammer Attacks, which is essentially a you know new class of micro architectural attacks which targets the DRAM. So, like we have seen like previous attacks which targets the SRAM or the cache memory, but this actually targets the DRAM which is present in our modern day processors.

(Refer Slide Time: 00:41)



So, the concepts that I will be covering are first we will take a look at the DRAM organization, we shall be trying to develop an idea of row hammers. Then we shall be trying to study the address mappings like the various address mapping from the virtual address to the physical address, and also like from the virtual address to the cache memory like how essentially we kind of access the cache sets and the corresponding slices in the cache. And finally, we shall take a look at the attack and also discuss a potential countermeasure which is essentially popular against row hammer attacks.

(Refer Slide Time: 01:12)



So, to start with here is a look at the DRAM organization. So, we these are very simplified view of how the DRAM looks like. So, you can see that the common form of the DRAM chip essentially has got several components and the components can be studied in this following or it is organized in this following fashion. So, on one hand you have got the DIMM. So, the DIMM essentially is shown by this green bar.

So, this stands for the Dual Inline Memory Module and this typically comprises of several ranks of memory. So, for example, here it has been shown that there are two ranks and each of these ranks intern corresponds of you know like several cells of the DRAM memories, which essentially called as banks. And therefore, the row comprises of several banks for example here there are eight banks shown and if you look into each of these banks, then it is basically a regular organization of cells DRAM of DRAM memories. And the each of these essentially are or can be realized potentially by a one transistor implementation where there is one transistor and there is one accompany capacitor which basically stores the content of that specific cell.

So, the DRAM essentially is a regular organization, which is essentially comprises of these components; that means, the cells, the banks, the rows and the DIMM and finally, the channel which basically connects it with the memory controller.

(Refer Slide Time: 02:44)

So, the idea is that what we will be studying in this particular attack is essentially or basically originates from understanding the physics inside of what happens in your bank in your DRAM memory. So, for example, each of these or if you go into the bank of the memory, then we will see that the DRAM cells or organized as rows and columns its regular organization of rows and columns and there is one access transistor. So, this is the transistor with basically kind of stores or essentially is enables to write and also read data from the cell, and you can see that there is a capacitor which basically stores the content of the specific cell.

Now, this access transistor is connected to a word line. So, this is your word line ok. So, which when on connects the capacitor to the bit line ok. So, this bit line so basically you can read or write the corresponding data through a row buffers, there is also an accompanying row buffer. So, the idea is that when this specific word line and bit line gets connected, then this particular transistor or the corresponding capacitor basically gets connected to the row buffer. So, the content of the entire row essentially is deposited into the row buffer and therefore, we can read the content from the row buffer. So, reading or writing is done through a row buffer which can hold charge for the entire row.

Therefore the entire row if I for example, want to access the row 2; then the content of row 2 is stored in the row buffer. So, the row buffers serves as something like a cache of your content; that means, you know like next time in you are accessing the row 2 then

you do not need to go and access row 2, but you can get that content quiet fast from the row buffer. So, that essentially improves the overall performance of the DRAM cell or the DRAM memory.

On the other hand if you are reading from another row for example, from row 1, then you essentially have to kind of evict that data from the row buffer and you have to basically put the data from row 1 to row buffer and then read the content from row buffer. So, there is something like a conflict which can happen in the row buffer. So, typically there are three steps when you are accessing this. So, here there is an opening row; that means, you basically enable the row, you read or write to the cells and then you close the row. So, these are like the three broad steps that would take place. So, as we know that in a one transistor model or in this kind of DRAM cells the charge in the cells will decay which time and therefore, this needs periodical refresh.

Otherwise right the content of the data gets essentially tampered and that would potentially lead errors or falls inside your DRAM cells. So, typically the refresh period that we use is you know 64 milliseconds; that means, every 64 milliseconds there is a refresh cycle which happens to kind of you like refresh the states of the corresponding cells.

(Refer Slide Time: 05:29)

What is Rowhammer?

Rowhammer is the term coined for disturbances in recent DRAM chips in which repeated row activation causes the DRAM cells to electrically interact with each other.

Repeated discharging and recharging of the cells of a row results in leakage of charge in the adjacent rows.

If repeated enough times, typically before the automatic refresh in the adjacent rows, causes flipping of bits - phenomenon termed as Rowhammer.

Rowhammer

swayam

So, what is row hammer? So, row hammer is essentially a bulk or you know like an exploit which has been found in the modern day DRAM chips, and it essentially states

from the fact that row hammer is the term which is coined for the disturbances in the recent DRAM chips, which repeated row activation causes. So, those are if you are repeatedly activating the rows then a specific kind of errors occurred which are called as row hammers or row hammer errors. So, this activation or repeated row activation causes the DRAM cells to electrically interact with each other, and this repeated discharging and recharging of the cells of a row results in leakage of charge in the adjacent rows.

So, if repeated enough times typically before the automatic refresh because there is a seeing with the refresh happens only in 64 milliseconds. So, if we are accessing this faster than that, then this repeated accesses can cause potential flipping of bits in the adjacent rows and this essentially is termed as the row hammer or essentially the row hammer bug. So, this instrument of doing the repeated access is essentially called a row hammer and we will see like how that can be used as a potential instrument to create faults in targeted secrets, which are located somewhere in our DRAM cells.

(Refer Slide Time: 06:49)

So, what is the underlying principle or let us see how is we can perform hammering by seeing a very high level code for doing the doing this operation.

So, here is an regular arrangement of the cells arranged in rows and columns, and this is the corresponding row buffer. So, what we are trying to do here is, basically we are trying to make a regular access to this components and we essentially the objective is

that we would like to create a fault inside one of the rows. So, what we basically try to do is encamp or shown here by this code snippet.

So, we can see here, we so we are trying to use these for example, these two to move instructions, essentially are trying to read from address X and read from access Y. So, the idea is that when you are reading from access X and when you are reading from Y. So, you are basically wanting to read from the content of X and Y.

So, note that one thing that we will always keep in mind throughout our discussion on the attack is that, the address when we are accessing a specific memory location then that content could be there in your potential cache memory. So, therefore, if you really want to kind of understand or observe the phenomenon what happens in your actual DRAM cell, we have to ensure that data is not available in the cache. So, we have to kind of evict the data from the cache.

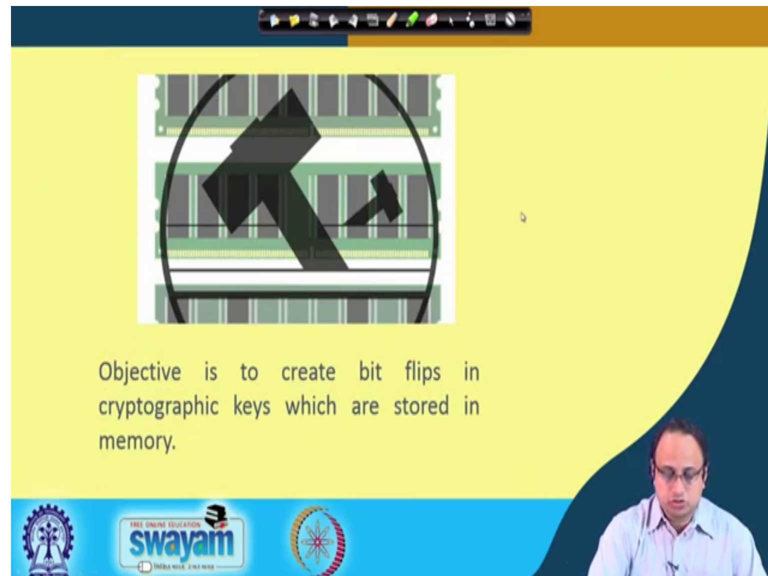
So, we can use dedicated instruction and shown here which is like clflush; which essentially evicts the data from all levels of the cache. For example, when we do a clflush X then the data is not present in the cache, similarly if I do the cl flush Y, then also that data is not available in the cache. So, these two things basically ensures that next time when I am accessing the data from the address locations X and Y, then I am indeed accessing the DRAM cells and not the cache memory.

So, when I am accessing for example, these two rows like X and Y. So, these are two different addresses and imagine that they get mapped into two different rows in a specific bank of the DRAM. Then the data right would be so the initially when I am accessing for example, this row then say the i minus 1th row then the content basically comes from the row buffer and therefore, the next time if I access the same row same row, then I get the data from row buffer.

So, in order to create the flip we basically as we know that we have to basically ensure that every time I have to access the corresponding row in the bank and not the row buffer. So, therefore, I need to ensure that I basically make the next access same maybe at address Y which is at a different row in the corresponding bank. So, therefore, right if this two accesses for example, the access to X and the access to Y happens very fast kind of faster than its refreshed cycle, then the adjacent row as shown here as the i th row shows or may show sudden bit flips.

For example the one can potentially get 0 and that essentially would mean that there is a suddenly a fault which happens in your adjacent row. So, we have seen that faults right essentially can be quite catastrophic to cryptographic implementations and also in general security. And therefore, right flipping bits in memory without really accessing them is really a serious threat that we need to consider.

(Refer Slide Time: 10:12)



So, therefore, right I mean of course, like modern day DRAM cells comes with several levels of protection, but we can see that in certain scenarios we are able to create such kind of bit flips and therefore, it seems like this instrument of row hammer is a quiet powerful tool, and we can potentially apply it with the objective to create bit flips in cryptographic keys which are stored in memory. Of course like it would not be easy, but it seems like we can potentially try to do that and that is something that we would like to study subsequently.

(Refer Slide Time: 10:45)

So, we have an instrument like the row hammer and we would like to apply it and we would like to study for so far what are the challenges and objectives of you know like this instrument that we have found.

So, the first thing we observe is that the secret is residing in some location in the cache memory, and also in some unknown location in the main memory kind of something that I do not know because I do not have a handle exactly where it is where the where it is located and where it gets mapped into the actual DR memory or even the cache memory.

So, the attacker having user level privileges in the system does not have the knowledge of this locations even in the last level cache. So, we will consider in the last level cache because typically the access is shared by several cores and apparently this makes the attack much more conducive. So, we will be considering the LLC cache and the DRAM, since these locations are decided by mapping so the physical address bits.

So, as an attacker I do not have knowledge about or direct knowledge about the corresponding physical address bits from which I can understand you know like the corresponding locations in the DRAM or even in the LLC. So, in order to perform the row hammer on the secret exponent the adversary first needs to identify the corresponding bank in DRAM in which the secret exponent resides that is the first objective that we have. And along with it if the adversary frequently queries the decryption oracle; that means, you know like for example, we will be assuming the threat

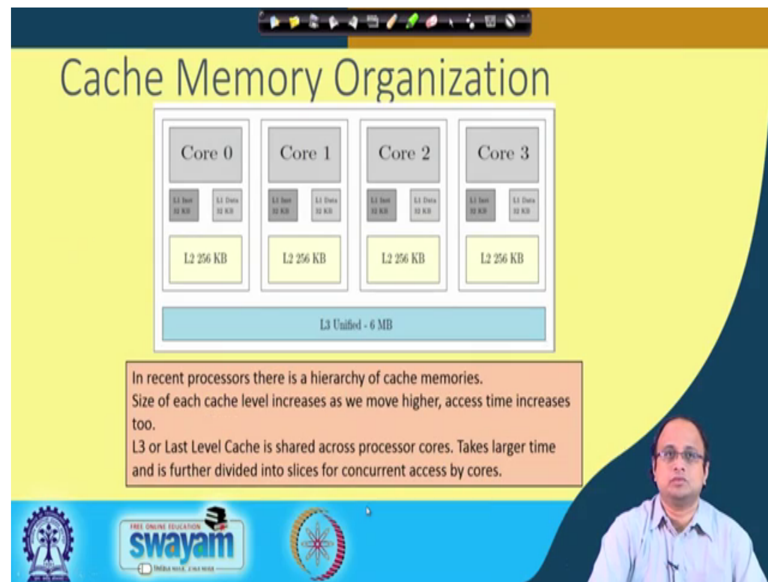
model where the adversity is kind of interacting with your target, basically is allowing the target or the victim to perform decryption with the secret key.

So, the idea is that, if the adversary frequently queries the decryption oracle with valid cipher text decryption process will perform exponensation with which will basically make an access to the secret exponent and that is what we want as an attacker. We want that the attacker again and again performs decryption with the same key and we would basically in that process or during that time try to understand where that secret is located in the DRAM cell and also even in the LLC cache.

So, but access requests as I said right are usually addressed from the cache memory because again and again if you allow the you know like the process to the victim to execute on the same content then that data would be available in the cache memory. And therefore, we will not be able to observe this phenomenon and exploit it. So, therefore, what we do is basically make an access; we basically adversary incorporates a spy which runs concurrently to the target process.

And then uses the timing side channels something as we have already seen previously in context to do when we discussed about cache attacks, what is called as prime and probe attacks and we basically kind of employ a prime and probe attack to understand the mapping into a channel, rank, bank where the secret has been correspondingly mapped to. We also would like to kind of use this spy to ensure that the data is always evicted from the cache; that means, we basically I want to know the corresponding mapping of the secret exponent into the last level cache. So, that even the spy right is making an access I mean the victim basically is always making an access to the actual DRAM cell and not to the cache memory.

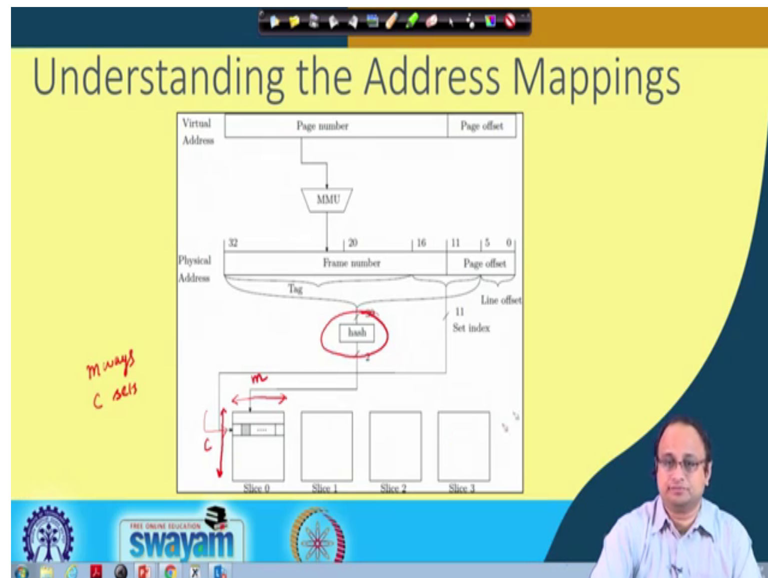
(Refer Slide Time: 14:12)



So, this is you know like a very simple organization of cache memory a very standard organization of the cache memory. So, for example, consider in our system we will be considering a four core architecture. So, we know that we have got private L1 caches we have got L2 caches and then there is a unified L3 cache that is essentially serving as the last level cache. So, the idea is that in recent processes there is a hierarchy of cache memories, the size of each cache level increases as we move higher.

So, you can see that the size progressively increases and the last level cache is shared across processor cores and it takes larger time and its further divided into slices for concurrent access by cores. So, targeting the last level cache is therefore, conducive because it is shared by across by several cores and the effect is more observable.

(Refer Slide Time: 15:10)



So, what we can observe here is that it is the first thing is that therefore, we have to understand the address mappings like from the virtual address to the physical address. So, let us take a quick look back to our architecture textbooks. So, here is a virtual address mapping. So, which is essentially divided into two parts one is the page offset and the other part is the page number. And then the memory management unit basically kind of kind of has got an mechanism of giving us the frame number and the page offset, which is basically combined to get the corresponding physical address. However, when you are accessing the cache memory for a four core machine, we will be essentially having four slices in the cache.

The idea is that the mapping from the physical address to the corresponding location in the. So, basically like this is an n way said as reduce cache. So, in our particular architecture we will be considering a 12 way set associative cache, that is the experimental setup that we will be considering, but in general it is a m way set cache and these are all the four potential slices.

So, each slice essentially has got several sets and every set. So, imagine that suppose there are c sets for example, and each of these sets essentially has got n ways ok. So, therefore, potentially right we can have n ways for every slice and there are c potential sets. So, these are my corresponding c sets and there are n ways in which the mapping can occur.

So, therefore, right I mean you can observe that there is a hash function and we will kind of elaborate about that subsequently this essentially has been unfired in some of the research papers and essentially potentially gives us a reverse engineering on this mapping because this may not be exactly documented in the available white papers. So, there are some prior research with basically tells us how this mapping takes place.

And as we can see that the set index here basically tells us what is the corresponding set index, but there are m ways. So, there are any of the m ways are potentially possible and where this particular address maps for example, the corresponding slice where it maps to is indicated by this hash function which takes a 30 bit input and generates a 2 bit output which is used to index any of the 4 potential slices. So, this is the corresponding mapping that we would like to understand and the cipher as much as necessary so.

(Refer Slide Time: 17:51)

So, therefore, right with this background let us take a look at the corresponding attack strategy. So, the first job right is basically to ensure that when the target of the victim basically executes on your unknown secret exponent, then every time it is accessing the secret exponent it is basically making an access to the DRAM and not to the cache. So, we have to evict it and because of that right we have to understand the eviction set, we have to understand the corresponding eviction set; that means, the footprint of the secret exponent or the cache memory.

In order to understand that, we will basically apply a technique which is called as prime and probe and that is something that we have already studied previously. So, the idea is that the adversary is oblivious of the virtual address space which is used by the decryption engine and thus involves a spy process to do that. So, the adversary basically kind of uses a spy and the spy essentially you know like so, basically this spy process uses a basically uses or employs this prime and probe cache excess methodology to identify the target sets. So, the idea is basically the spy primes are corresponding area in the cache memory and subsequently comes back after the execution of the victim and probes its own access.

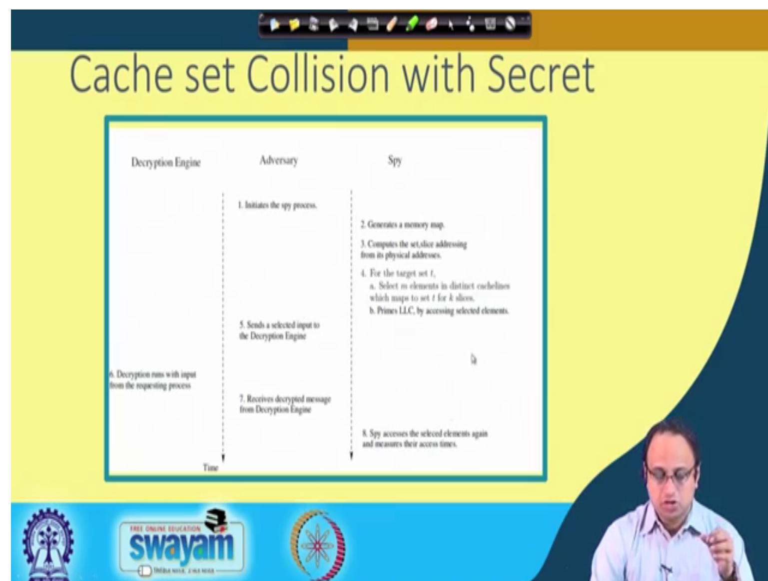
If the individual access time takes a significantly large amount of time, then it basically kind of you know like assumes or guesses that those areas has been evicted by the victim process and that is why it is taking more time and with that right basically understands this mapping and understands eventually the eviction set which is essentially required to ensure that or this understanding is important to ensure that, every time the secret exponent is kind of access subsequently when we actually launch the attack that access should be done from the DRAM and not from the s ram or the cache memory.

So, the idea is that the spy process targets the last level cache as we have discussed. Since it is shared with all codes of the system the spy initially allocates a set of data we elements and consults its own page map to obtain the corresponding physical addresses for each element. So, in this attack right we are basically assuming that the spy essentially has got the user level access to the two page map, and with that essentially it can know the its own physical addresses ok. How it must be kept in mind that because of this potential attacks subsequent to Linux kernel version 4 this access to page map has been made restricted. However, there maybe you know like several systems where still these are enabled and more importantly right in embedded processes right we may still have access to this kind of page maps at user levels and which could potentially a security threat.

Moreover right there could be other ways of circumventing this, but at least right for this within this scope of this cores we will be restricting and assuming an access two page map; that means, the spy right can understand the corresponding physical addresses of its own access patterns. So, for example, if we makes a memory map, then it knows the corresponding physical address of the accesses which it is making itself.

So, there is essentially you know like a quick look about a you know like a document from www dot kernel dot org which basically tells that since Linux 4.0. If you are accessing this without the user without the root privilege, then you would not get the page map access. And the reason is the information about this is restricted because this has been exploited in row hammer vulnerabilities and therefore, this is a serious threat to be considered.

(Refer Slide Time: 21:38)



So, here is the basic broad strategy behind the attack. So, the cache set collision with the secret. So, we basically want to kind of understand the eviction set and we can understand this in a step by step process which is illustrated in this diagram.

So, the adversary basically initiates the spy process and the spy process basically generates a memory map. So, it does a memory map and it computes the sets slice sets slice addressing from its physical address. So, as I said that it knows its corresponding physical address by using the re page map. And for the target set t for example, you basically fixes the target set t. So, the target set t as we have already seen it can be found out from its physical address and right I mean for every target set c then so I set t there are n possible ways because it is a n way set cache. So, it selects n elements in distinct cache lines which maps to a set t for all the k possible slices because there are k slices that is what we are assuming. And it primes the LLC by accessing this selected elements.

So, basically primes means it kind of fills up the cache memory with these potential locations and then it basically sends a selected input to the decryption engine and allows the target or the victim to decrypt. So, note that if the when the corresponding cipher text is returned back to the adversary because the adversary is also interacting with the decryption engine, and that is quite a practical threat model. Because any adversary can access or you know like can interact with a secret or with essentially a decryption engine. And when it receives the decryption or decrypted message from the from the decryption engine it basically asks the spy to get or to start probing because it has already done the priming operation now it will basically start probing.

So, the probing means the spy will access the selective elements again and for every access we will measure its time. The idea is that if this time is more than the threshold then it will basically kind of conjecture or assume is that, the decryption engine probably has come and evicted those things and that is why I am taking more time and through that it basically understands the footprint and from there the eviction set for the decryption exponent.

(Refer Slide Time: 24:04)

So, note that one thing should be kept in mind that this mapping from the virtual address to the corresponding physical address can change every time. So, therefore, right what we are studying right now is a methodology, but that does not mean when we again start or restart this process the mapping will be the same. So, therefore, we have to do as we

will see later on when we do the attack we have to do the attack all at once, we have to do the attack in situ ok. But this basically tells us how the mapping can be deciphered and this basically we will use as a subsequently as an oracle in my broader attack which we will discuss subsequent to this.

So, this basically kind of tells us about the reverse engineering of the cache slice selection. If the target system is having k processor cores then the LLC has got k slices as we have already seen, each as slice has a has got c cache sets and each set being m way associative or there is an m way associatively. So; that means, that if the cache line size is of b bytes like if the cache size is of b bytes for example, 64 bytes, then we will be using the least significant log 64 base 2 or log b base 2 bits of the physical address as a index within the cache line. And again we will be for getting the set number we will be using log c base 2 to indicate or determine the cache set number.

And because of associatively n such cache lines having identical log c base 2 bits reside in the same set. Now we already discussed about the hash function which basically is already you know like established in this two prior papers which tells about how to perform the reverse engineering to know or is or obtained the corresponding slice number from the physical address.

(Refer Slide Time: 25:58)

Selecting elements belonging to Eviction set

To precisely control the eviction of existing cache lines from set t, spy runs a selection algorithm to select an *eviction set* of $m * k$ elements belonging to each set t.

- Thus the selection algorithm selects elements of distinct cache lines for each of the k cache slices such that physical addresses maps to the same set t.
- In addition, each set of a slice is m way associative, the selection algorithm selects m elements corresponding to each k cache slice belonging to set t.
- The spy accesses each of these $m * k$ selected memory elements repeatedly to ensure that the cache replacement policy has evicted the existing cache lines.

So, with this reverse engineering in bind what we do is to precisely control the eviction of existing cache lines from the set t, the spy runs the selection algorithm to select an eviction set of m into k elements belonging to each set t then the selection algorithm.

So, basically selects elements of distinct cache lines for each of the k cache slices; such that the physical addresses maps to the same set t. In addition each set of a slice is m way associative and therefore, the selection algorithm selects m elements corresponding to each k cache slice belonging to the set t. The spy accesses each of these m into k selected memory elements repeatedly to ensure that the cache replacement policy has evicted the existing cache lines.

So, the idea is that right now if you basically observed then you are in a scenario where you have basically been. So, if you have basically target a specific set there are m possible ways over here right. So, this essentially is a specific slice of the cache suppose the slash i slice i for example, there are m possible ways and there are k such slices. So, suppose there are k slices. So, therefore, if we if you kind of understand a or fix the corresponding set, then totally there are m k possible addresses or m into k possible addresses, there are m ways part slice and there are k slices.

So, totally there are m k possible addresses. So, these are the you know like for every target say t therefore, the spy basically runs a selection algorithm and therefore, it basically probes these m k selected memory elements and tries to understand which set corresponds to the secret exponent.

So, the idea is that it basically will probe for all the corresponding c sets that are there. So, remember that there are potential is c sets that we are considering and for every set there will be m k possible locations. So, it will basically probe these locations and try to understand which set among the c sets are corresponding to the secret exponent ok. So, this essentially implies that, with this setting what we essentially can do right now is we can basically launch the prime and probe analysis.

(Refer Slide Time: 28:30)

Prime + Probe

1. The spy primes the target Set t and becomes idle.
2. The adversary sends the chosen ciphertext for decryption and waits till the decryption engine sends back the message.
3. If cache sets used by the decryption is same as spy, then the cache lines primed by the spy process gets evicted during the decryption.
4. Adversary signals the spy to start probing and timing measurements are noted.
5. Spy process accesses each of the selected m elements of eviction set t for all slices and time to access each of these elements are observed.
6. When the spy again accesses the same elements, if it takes longer time to access then we conclude that the cache set has been accessed by the decryption as well.

So, the spy basically primes the target set t. So, that is your specific target set t out of the c possible sets and becomes ideal.

So, prime means it basically fills up that fills up the cache with those m k possible addresses. The adversary sends the chosen ciphertext for decryption and waits till the decryption engine sends back the corresponding message after decrypting. So, note that the objective of the attacker is not to get the message, but to get the secret exponent because that is more important. If cache sets used by the decryption is same as the spy, then the cache lines primed by the spy process gets evicted during the decryption and the adversary signals the spy now to start probing.

So, it basically starts to probe at those mk locations and the spy process accesses each of the selected n elements of the eviction set t for all the k slices and the time to access each of these elements are observed. When the spy again accesses the same elements if it takes longer time to access then we can conclude that the cache set has been access by the decryption and therefore, it takes more time.

(Refer Slide Time: 29:44)

Experimental Setup

1. We target an 1024 bit RSA implementation using square and multiply as the underlying exponentiation algorithm using GNU-MP big integer library (version number 2:5.0.2+dfsg-2).
2. The experiments are performed on Intel Core i5-3470 processor of Intel Ivy Bridge micro-architecture running Ubuntu 12.04 LTS with the kernel version of 3.2.0-79-generic.
3. The Linux kernel version for our experimental setup being older than version 4.0, we did not require administrator privileges to perform the entire attack.

So, one can you know like try to make this attack more interesting and can try to go into the slice also, because remember that we also have an understanding about how the slice has been selected and we will see subsequent to this right that how we can make the attack even more concentrated or more focused by getting into the slice, where we can reduce the possible sets or eviction set from m into k to only m values because now we basically are considering only specific slice.

So, for illustrating this attack we will basically considering in our subsequent class on a 1024 bit RSA implementation, which is essentially using square and multiply as the underlying exponential algorithm using GNU-MP big integer library and this is the corresponding version, and the experiments will illustrated on Intel Core i5 processor which has got an Intel Ivy Bridge micro architecture which is running an Ubuntu 12.04 LTS with the Kernel version of 3.2. Note that here we can essentially get unrestricted page map ok. And the Linux as I already mentioned the Linux kernel version for our experimental set up being older than version 4.0 we do not require administrator privileges to perform the entire attack, but the exact description of the attack we will see in the next class.

Thanks for your attention.