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

Lecture - 60
Microarchitectural Attacks: Part 3 Row Hammer Attacks (Contd.)

So, welcome back to this class on Hardware Security. So, we shall be continuing our discussions on Row Hammer Attacks.

(Refer Slide Time: 00:22).

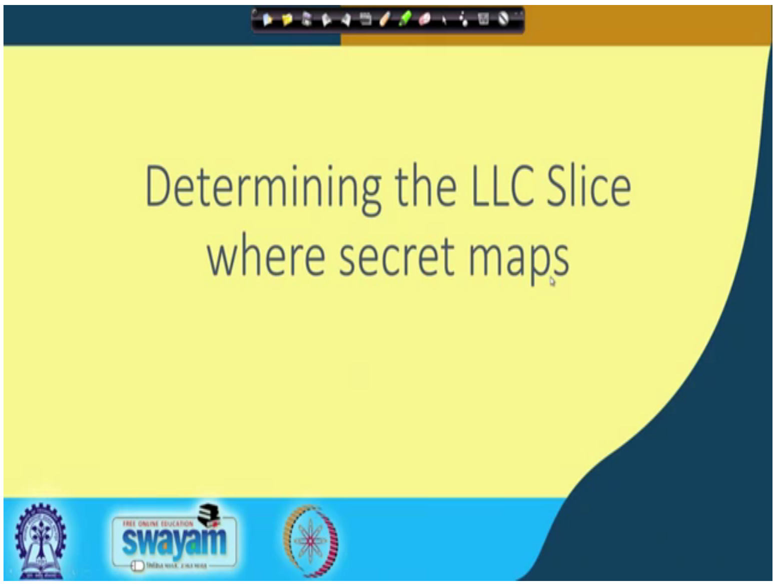
CONCEPTS COVERED

Concepts Covered:

- ❑The Attack
- ❑A Countermeasure

So, we shall we already kind of seen the background behind the attack and today we shall be seeing the actual attack and also discuss a potential countermeasure.

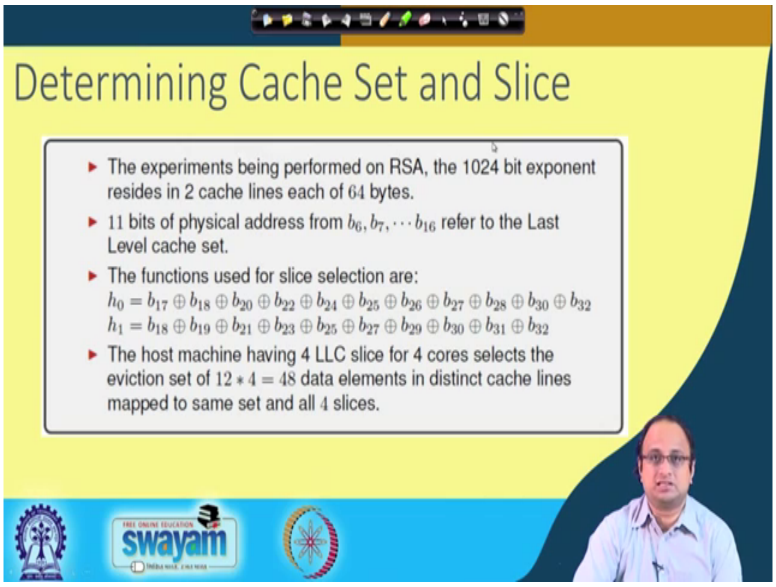
(Refer Slide Time: 00:29)



Determining the LLC Slice where secret maps

So, where we stopped in the last class, where basically determining the LLC slice where the secret maps. So, we saw how prime and probe can be utilized in this context.

(Refer Slide Time: 00:41)



Determining Cache Set and Slice

- The experiments being performed on RSA, the 1024 bit exponent resides in 2 cache lines each of 64 bytes.
- 11 bits of physical address from $b_6, b_7, \cdots b_{16}$ refer to the Last Level cache set.
- The functions used for slice selection are:
  $h_0 = b_{17} \oplus b_{18} \oplus b_{20} \oplus b_{22} \oplus b_{24} \oplus b_{25} \oplus b_{26} \oplus b_{27} \oplus b_{28} \oplus b_{30} \oplus b_{32}$
  $h_1 = b_{18} \oplus b_{19} \oplus b_{21} \oplus b_{23} \oplus b_{25} \oplus b_{27} \oplus b_{29} \oplus b_{30} \oplus b_{31} \oplus b_{32}$
- The host machine having 4 LLC slice for 4 cores selects the eviction set of $12 * 4 = 48$ data elements in distinct cache lines mapped to same set and all 4 slices.

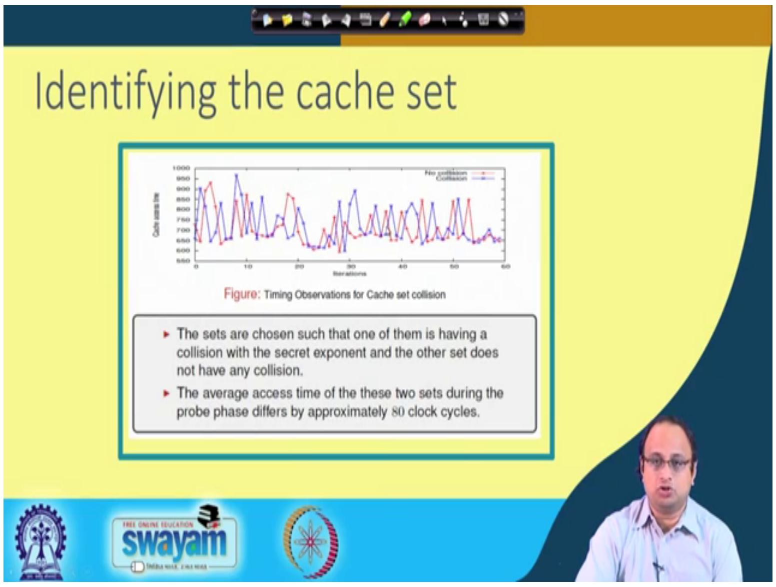
So, in particular right we already discuss that we basically; so, the experimental setup that we are considering here is essentially an RSA implementation with 1024 bit exponent, which basically can reside in 2 cache lines each of 64 bytes. So, you can see that totally it is 124 bytes or bits, so it can be organized or it basically takes or occupies 2 cache lines where each of them are of 64 bytes, 64 into 8 into 2.

So, so then 11 bits of the physical address is referred to the last level cache set and essentially gets your corresponding cache set in your and we are considering 4 cores; that means, we have a mapping to get 4 cores and there are 4 corresponding slices in the cache memory. So, the corresponding hash functions which essentially has been reverse engineered in a in a pair work which I mentioned in the last class, essentially are shown by these two equations.

So, you can see it is a simple linear equation which apparently gives you the corresponding mapping of the from the physical address bits two bits h 0 and h 1 through which you can basically index this 4 LLC cache slices. So, now, if so previously right we had discussed about that we were is basically doing an attack on or trying to basically fix the corresponding cache set, where there are mk possible values right, for every slice there are m possible ways and there are cache slices.

So, putting the values here m is 12 because there is (Refer Time: 02:17) that we considered is 12 and there are 4 slices, so there are 12 into 4 48 data elements. So, we basically can the moment you fix a cache set there are 48 possible addresses which we used to prime and probe.

(Refer Slide Time: 02:33)



Identifying the cache set

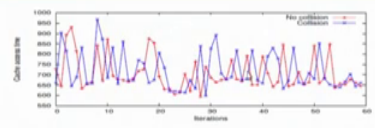


Figure: Timing Observations for Cache set collision

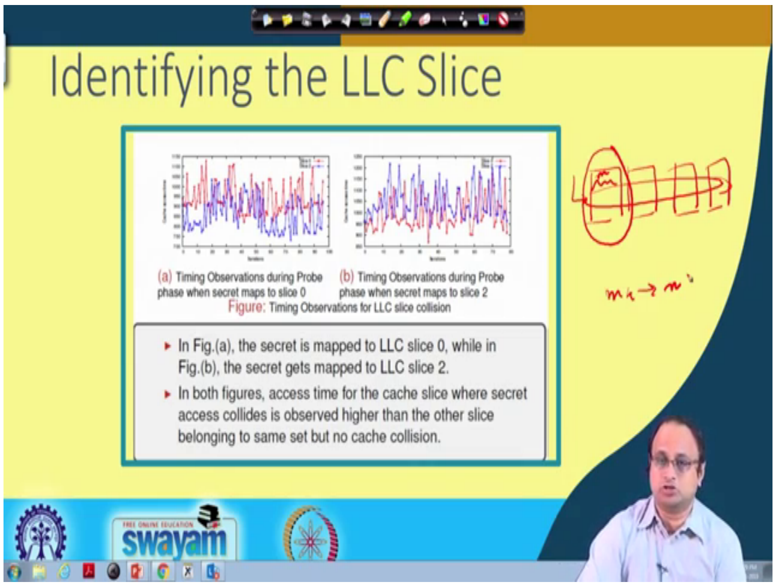
- The sets are chosen such that one of them is having a collision with the secret exponent and the other set does not have any collision.
- The average access time of the these two sets during the probe phase differs by approximately 80 clock cycles.

So, so here is an experiment to show that in it works. So, for example, you see that there are two graphs which has been shown over here, one which corresponds to the collision the for example, the blue which corresponds to here collision. That means, the corresponding cache set which you are basically using two prime and probe is basically colliding with your secret exponent.

And you can see that if there is an collision; that means, right that is essentially, so when the spy comes back and probes it basically expects that there should be an increase in the average time and that is what you can observe here. Since there is a collision you find that the average cache access time has been increased and therefore, the spy is able to understand the correct cache set, where the secret exponent is getting mapped to.

So, the sets are chosen such that one of them is having a collision with the secret exponent and the other set does not have any collision. The average access time of these two sets during the probe phase when you are coming and probing, you are seeing a differentiating by approximately by 80 clock cycles. So, we shows that there is a significant discrepancy between these two timings and basic any kind of leaks information about the LLC mapping of your secret exponent. But remember that at this point we basically have kind of still 12 into 8 48 possible address locations, 12 into 4 48 possible address locations.

(Refer Slide Time: 04:06)



Identifying the LLC Slice

(a) Timing Observations during Probe phase when secret maps to slice 0 (b) Timing Observations during Probe phase when secret maps to slice 2

Figure: Timing Observations for LLC slice collision

- In Fig.(a), the secret is mapped to LLC slice 0, while in Fig.(b), the secret gets mapped to LLC slice 2.
- In both figures, access time for the cache slice where secret access collides is observed higher than the other slice belonging to same set but no cache collision.

You can actually make this more concentrated by getting in the slice. So, you can use those two hash functions, to basically also kind of get mapped into not only m into k, but get into one slice so; that means, in one slice for cache set there are m possible ways. So, therefore, here is an example where you are considering 2 slices, so just to understand that whether you are correctly understanding the slice for example.

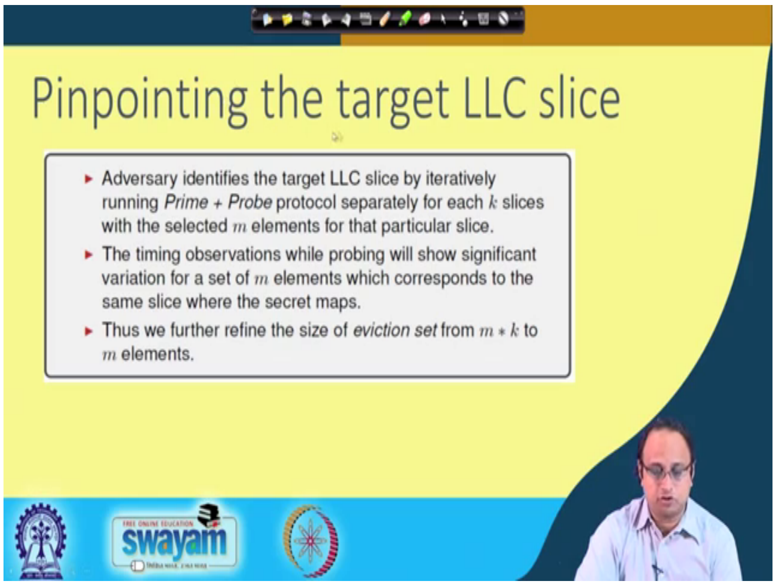
So, for example, in this case in figure a the secret is mapped into LLC slice 0, while in figure b the secret gets mapped into LLC slice 2 and you can see that when it is essentially getting mapped into slice 0, then slice 0 is indeed we are taking more time corresponding to slice 2.

Whereas, in this case the secret gets mapped into the LLC slice 2 and therefore, the probing phase indeed tells that the mapping for slice 2 is taking more time and therefore, you are correctly able to understand the slice and that in the way says that the reverse engineering for the hash functions for the slice computations are working as expected.

So, in both the figures the access time for the cache slice where secret access is collide is observed higher than the other slice belonging to the same set, but for no cache collision. So; that means, right previously you had an ambiguity about the, you know like when you are considering or in the previous case we had 4 slices.

So, for a given cache set right we had all these things in my eviction set and now if you can understand the slice also then you have just possible m possible ambiguities. So, it from m k you can reduce the ambiguity to m possible options.

(Refer Slide Time: 05:53)

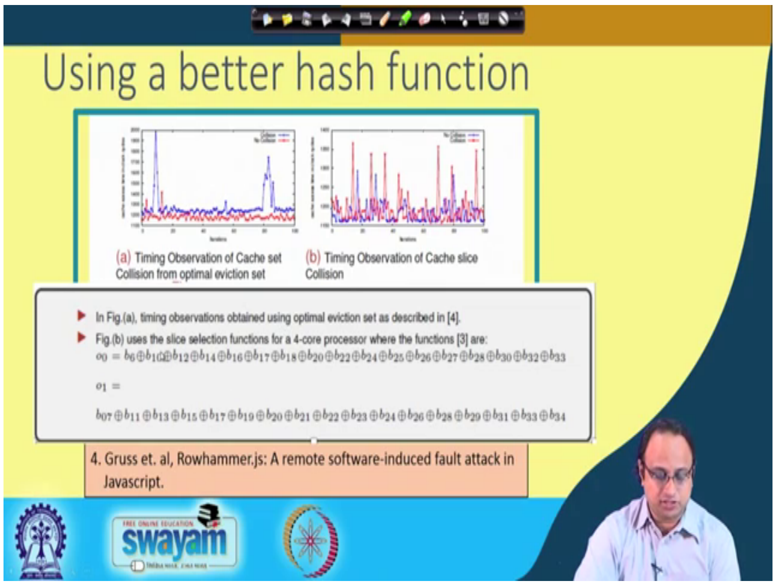


Pinpointing the target LLC slice

- Adversary identifies the target LLC slice by iteratively running *Prime + Probe* protocol separately for each k slices with the selected m elements for that particular slice.
- The timing observations while probing will show significant variation for a set of m elements which corresponds to the same slice where the secret maps.
- Thus we further refine the size of *eviction set* from $m * k$ to m elements.

So therefore, right I mean what you can do is improve it; so therefore, you can pinpoint the target LLC slice. And the adversary identifies target LLC slice by iteratively running prime and probe protocol separately for each k slices with the selected m elements for that particular slice. The timing observations while probing will show significant variation for a set of m elements which corresponds to the same slice where the secret maps and thus we further define the size of eviction set from m into k to only m possibilities.

(Refer Slide Time: 06:24)



Using a better hash function

(a) Timing Observation of Cache set Collision from optimal eviction set

(b) Timing Observation of Cache slice Collision

- In Fig.(a), timing observations obtained using optimal eviction set as described in [4].
- Fig.(b) uses the slice selection functions for a 4-core processor where the functions [3] are:
  $o_0 = b_6 \oplus b_{10} \oplus b_{12} \oplus b_{14} \oplus b_{16} \oplus b_{17} \oplus b_{18} \oplus b_{20} \oplus b_{22} \oplus b_{24} \oplus b_{25} \oplus b_{26} \oplus b_{27} \oplus b_{28} \oplus b_{30} \oplus b_{32} \oplus b_{33}$

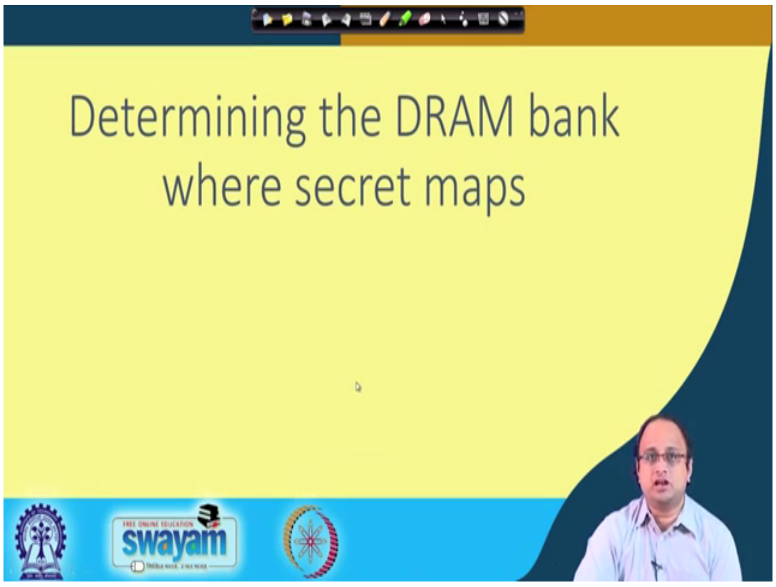
  $o_1 =$

  $b_{07} \oplus b_{11} \oplus b_{13} \oplus b_{15} \oplus b_{17} \oplus b_{19} \oplus b_{20} \oplus b_{21} \oplus b_{22} \oplus b_{23} \oplus b_{24} \oplus b_{26} \oplus b_{28} \oplus b_{29} \oplus b_{31} \oplus b_{33} \oplus b_{34}$

4. Gruss et. al, Rowhammer.js: A remote software-induced fault attack in Javascript.

So, here is another alternative hash function which was presented in a following work and if you use it right then apparently you can get a better separation, but so; that means, right you can actually use potentially few options and essentially it works either or. So, essentially you can take any one of them as a possible representation of the corresponding mapping.

(Refer Slide Time: 06:53)



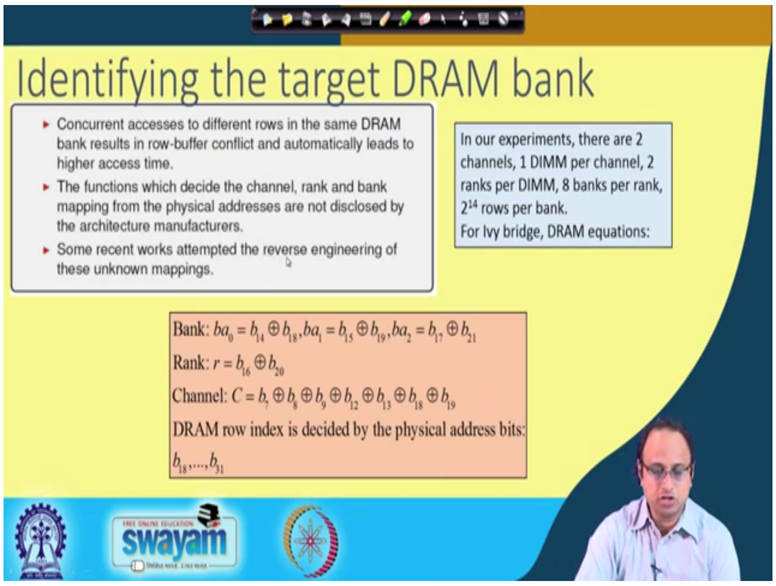
Determining the DRAM bank where secret maps

So, now you have to basically get into the actual attack, but for the actual attack you also need to kind of determine the DRAM bank where the secret maps. So, the objective of the eviction determination was for the cache set eviction detection was basically to ensure that the decryption when you are doing the decryption.

Then you are basically accessing the DRAM, but now you have to do the actual row hammer which means you have to basically ensure that when you start accessing again and again in a very repeated manner, then you basically access the bank where the secret essentially getting is getting map to.

So, now; so, in the first phase of the attack you basically kind of restrict the attacker to go to the DRAM bank, but now you also have to go to the same DRAM bank and do the row hammering operation. So, you need to do this reverse engineering.

(Refer Slide Time: 07:40)



Identifying the target DRAM bank

- Concurrent accesses to different rows in the same DRAM bank results in row-buffer conflict and automatically leads to higher access time.
- The functions which decide the channel, rank and bank mapping from the physical addresses are not disclosed by the architecture manufacturers.
- Some recent works attempted the reverse engineering of these unknown mappings.

In our experiments, there are 2 channels, 1 DIMM per channel, 2 ranks per DIMM, 8 banks per rank, 2^{14} rows per bank.
For Ivy bridge, DRAM equations:

Bank: $ba_0 = b_{14} \oplus b_{18}, ba_1 = b_{15} \oplus b_{19}, ba_2 = b_{17} \oplus b_{21}$
Rank: $r = b_{16} \oplus b_{20}$
Channel: $C = b_7 \oplus b_8 \oplus b_9 \oplus b_{12} \oplus b_{13} \oplus b_{18} \oplus b_{19}$
DRAM row index is decided by the physical address bits: $b_{18},\dots,b_{31}$

So, therefore, in order to identify the target dram bank. So, we will again use a pair result which is essentially shown in some of these equations like from again from the physical address how to get the bank number, how to get the rank number and how to get the channel number. In particular you see that this is the corresponding way of how to get the bank number.

So, in our experiment there are 2 channels, there is 1 DIMM per channel, there are 2 ranks per DIMM and there 8 banks per rank. So, we basically finally, want to get into the bank right and since there are 8 banks you can observe that number of bits which you have for the banks are 3 ba 0, ba 1 and ba 2. So, therefore, the if you know the physical address you can calculate the value of ba 0, ba 1 and ba 2 and you can calculate the corresponding bank numbers for those physical addresses.

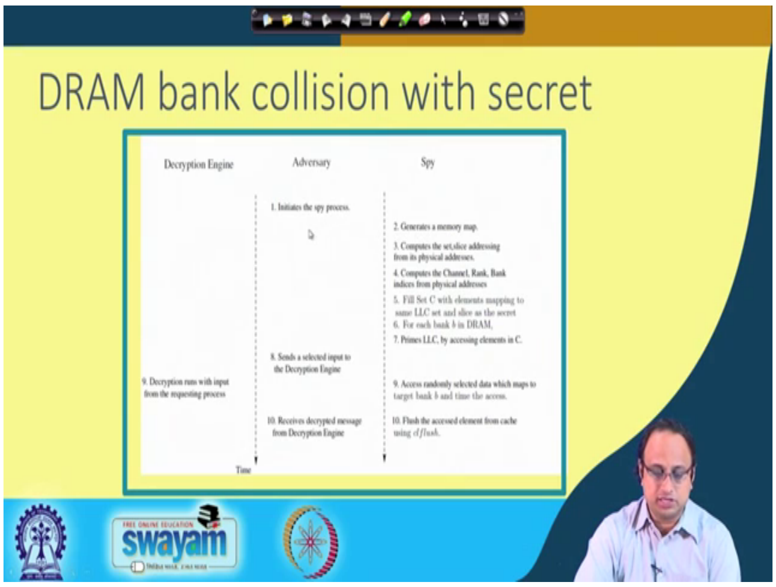
So, the objective is that the concurrent, so you want to basically inflict this row hammering or do the row hammering. So, you have to therefore, concurrently access basically what you want to do is, you want to develop or do concurrent access to the different rows in the same dram bank and this will result in a row-buffer conflict as we discussed, as we have remember the old code that we discussed when we basically where addressing x and y.

We wanted to access different rows in the bank, the objective is we wanted to kind of evict the data from the row-buffer. So, that rather than accessing data from the row-buffer I access the data from the actual rows. So, this is called as a row-buffered conflict and this will automatically lead to higher access time because if I get the data from the row-buffer that will take more time.

So, we again use this timing difference to understand, that whether we have been able to successfully evict the data from the row-buffer and we basically kind of access data from a different location in the bank or different row in the bank. Now the functions which decide the channel rank and bank map or bank making from the physical address are not disclosed typically, but as I said that we are used these equations which have been kind of published in some prior research works.

And also just to mention here that in a bank there are apparently something like 2 power of 14 rows. So, it is difficult for us try to get the exact row number in the bank. So, we basically do not have an handle to get into the row. So, we can basically get to the bank, but for you know like accessing or getting into the corresponding adjacent rows we basically have to do repeated attempts and assume that few of them will be successful.

(Refer Slide Time: 10:17)



DRAM bank collision with secret

Decryption Engine | Adversary | Spy

1. Initiates the spy process.
2. Generates a memory map.
3. Computes the set,slice addressing from its physical addresses.
4. Computes the Channel, Rank, Bank indices from physical addresses
5. Fill Set C with elements mapping to same LLC set and slice as the secret
6. For each bank b in DRAM,
7. Primes LLC, by accessing elements in C.
8. Sends a selected input to the Decryption Engine
9. Decryption runs with input from the requesting process
9. Access randomly selected data which maps to target bank b and time the access.
10. Receives decrypted message from Decryption Engine
10. Flush the accessed element from cache using $clflush$.

So, this is the complete elaboration the attack, so in this elaboration right we see that there is an adversary which initiates the spy process and as I said that you have to again decide for that eviction said that you know you have to do it every time in situ because the mapping from the virtual address to the physical address will basically change when the moment you basically run a different process.

So, therefore, you generate the memory map and you compute the sets slice addressing from the physical addresses, you compute the channel, the rank, bank indices different from the physical addresses again by using the equations that we just now discussed. And then we basically calculate the eviction set C and then you fill this or prime this with elements mapping to the same LLC set and slice as a secret.

And the idea is that for each bank b in the DRAM for each bank; b each bank. So, basically as I said that you now from your physical, so which you have basically obtained by doing page map. You basically can calculate the corresponding bank number using the equations ba 0, ba 1 and ba; 2 ba 0 and ba 1 and ba 2, you can get the corresponding bank numbers there are 8 impossible banks. So, you basically start targeting one of the bank for example, say the bank 0.

So, in bank 0 if we target bank 0 in that DRAM, you basically prime the LLC by accessing this elements in the c, so, basically now you start that actual attack. So, now after you have done this homework you may the moment you start you basically start your attack by fixing a corresponding bank numbers say bank 0 and then you basically prime the last level cache by accessing elements in your eviction set.

And then you basically the adversary or allow the adversary to or the adversary basically sells or triggers the description and note that since you already understood the eviction set previously, when the secret exponent runs since you have you are staying in the same eviction set or in the same footprint of the of the cache memory. Then that would imply that this location or this access has to be done from also it we are basically trying and sure that this essentially is not obtained from the cache, but is done from the actual DRAM ok.

So, therefore, right you basically do a decryption runs basically decryption runs it is accessing this two lines as we have discussed, but this access is done not from cache, but from DRAM because this address space or this eviction set is conflicting with my spy. The spy has ensured that I have kind of evicted that area, the address from the cache and therefore, when the decryption is running or executing that content is not available in the cache

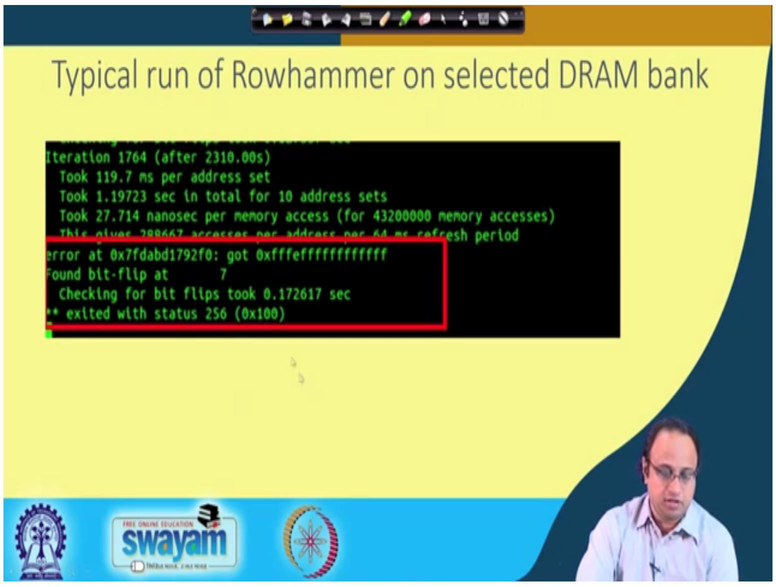
So, now when you basically come to you know like the moment the decryption explain the after a decryption basically runs. So, in parallel right or in conjunction you basically access randomly selected data which basically maps to the target bank b. For example the bank 0 and you basically time your access, so objective is that, now you know that the decryption is accessing a specific bank and you want that ah as a spy when you are accessing some locations you are also interested to access in the same back ok, but you also interested not to access the same row, but maybe some adjacent row ok.

So, how do you know that? So, the idea is that if you access the same row, then you would get the data from the same row-buffered and therefore, right you will basically take less time. So, that is why right you would kind of time your access because you can do that as a spy, you can time your own access. And you assume that if there is a more time requirement, then you basically are getting data from the row-buffered, but what you want or what you are interested is in the accesses which basically takes more time because that essentially is implied that you are accessing in adjusted rows.

So, and also right you after you do this access every time you would apply you know, you would like to flush or because the moment you make an access the data comes to the cache and you do not want to access the cache because you are interested to go in to the DRAM. So, you apply a specific instructions is called CL flush to ensure that this data is evicted from the cache and again right there will be the access, you are accessing into the actual DRAM bank and not the cash payment.

And finally, right you get the decrypted message from the decryption engine and this process is repeated again and again until and unless you are able to create the fault or the error.

(Refer Slide Time: 15:13)



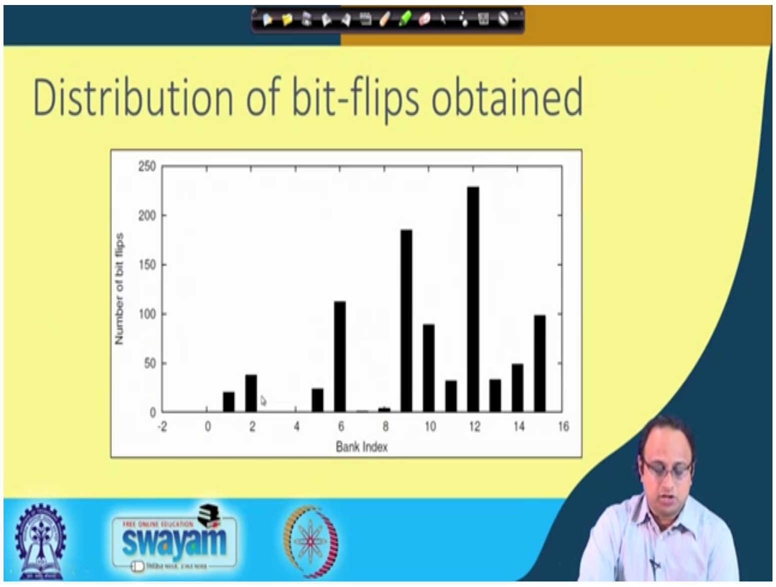
Typical run of Rowhammer on selected DRAM bank

```
Iteration 1764 (after 2310.00s)
  Took 119.7 ms per address set
  Took 1.19723 sec in total for 10 address sets
  Took 27.714 nanosec per memory access (for 43200000 memory accesses)
  This gives 288667 accesses per address per 64 ms refresh period
error at 0x7fdabd1792f0: got 0xfffeffffffffffff
Found bit-flip at       7
  Checking for bit flips took 0.172617 sec
** exited with status 256 (0x100)
```

So, after repeated runs for example, this is the short sets that it took some amount of time, but you can see that there is an error which has been reported in a specific bank for example, this has been found in bank 7, where there is a bit flip for example, initial everything was all f; that means, the entire content was all one, but suddenly there is a bit flip for which this f has become e.

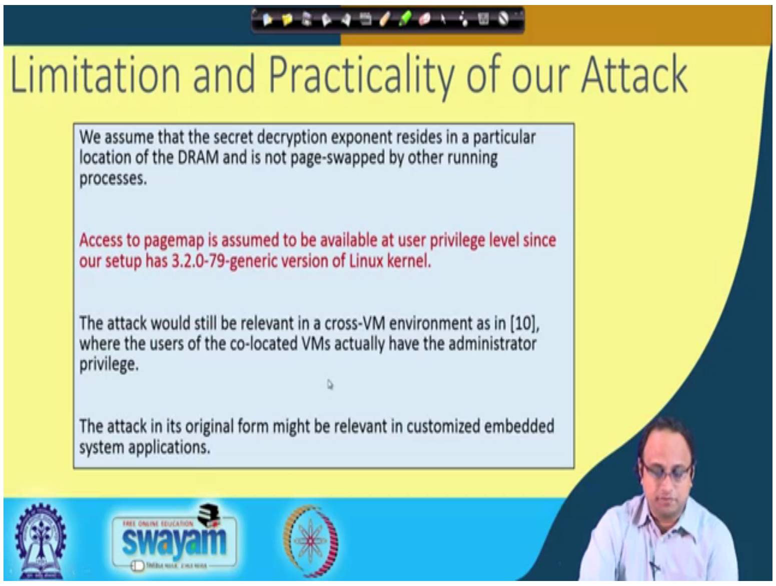
So, we know that if we are able to create one single flip then that essentially has got significant consequences if the security of cryptosystems whether it is AES or whether it is RSI or whether it is any other cryptosystems. And there right this flip right can be potentially exploited in doing or launching a fault attack for example.

(Refer Slide Time: 16:01)


Distribution of bit-flips obtained

So, here is for example, the distribution of bit flips that was obtained in our experiments. So, we have got like from 0 to several banks and we have we observe the bank indices and you can see that this is programmable codes the number of bit flips. So, you can see that some of the banks are not really prone to this kind of flips, but some of the banks are really prone to the kind of bit flips where the incidence of bit flips is more probable compare to the other. So, this is an interesting statistics that which can be observed.

(Refer Slide Time: 16:28)


Limitation and Practicality of our Attack

We assume that the secret decryption exponent resides in a particular location of the DRAM and is not page-swapped by other running processes.

Access to pagemap is assumed to be available at user privilege level since our setup has 3.2.0-79-generic version of Linux kernel.

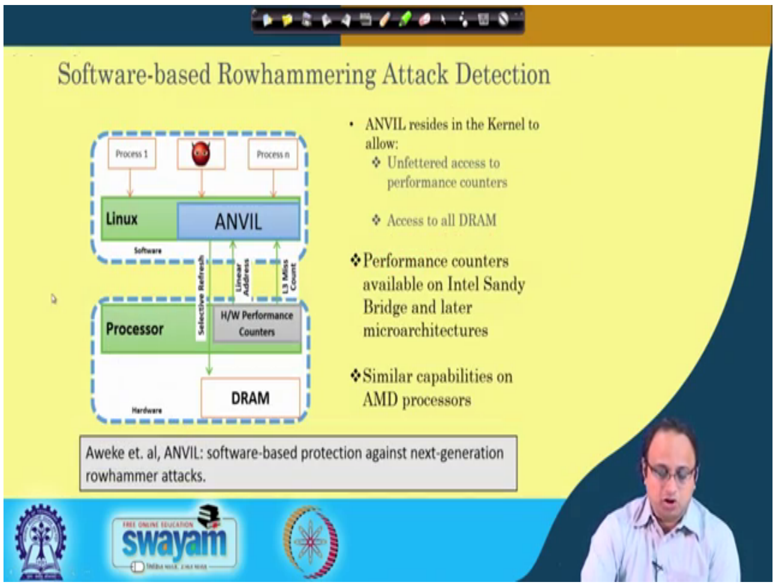
The attack would still be relevant in a cross-VM environment as in [10], where the users of the co-located VMs actually have the administrator privilege.

The attack in its original form might be relevant in customized embedded system applications.

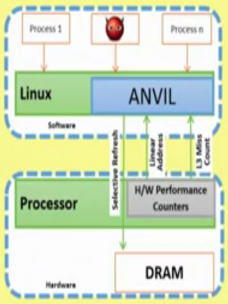
So, if we discuss about the attack, so this attack essentially shows that it basically assumes that the secret decryption exponent resides in a particular location of the DRAM and is not page swapped by other running process the mean that is not removed away to the disc for example. And so we can basically this seems to be a quiet practical assumption in several settings and the access to the page-map as I said is not available at user privilege, but still in several scenarios you may still find that his enabled there may be other ways of getting handle on the physical address as well.

And also maybe in some other settings right for example, in a cross VM environment right I mean where the users of the co located VMs actually have administrator or privilege you can potentially still use this page map and can if we still try to sort of conceptualize an attack. And the attacking is original form might be relevant in customize embedded system of applications where substrates can still be prevalent.

(Refer Slide Time: 17:34)



Software-based Rowhammering Attack Detection



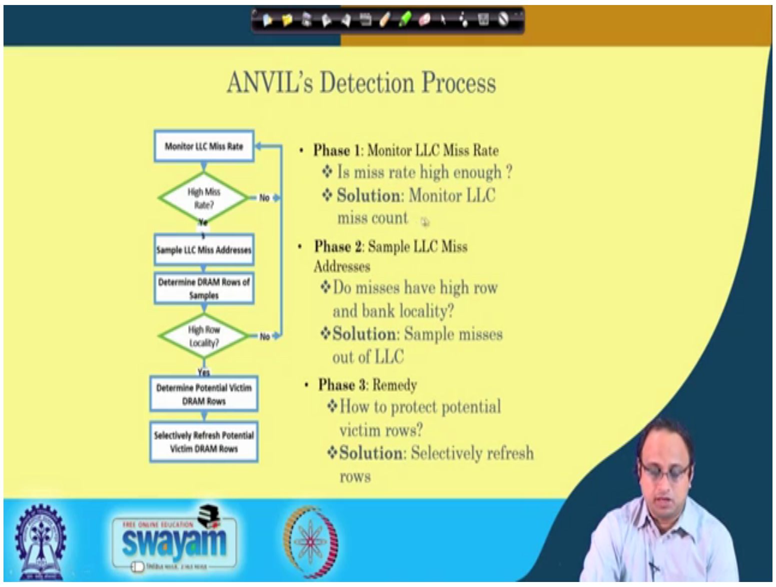
- ANVIL resides in the Kernel to allow:
  - Unfettered access to performance counters
  - Access to all DRAM
- Performance counters available on Intel Sandy Bridge and later microarchitectures
- Similar capabilities on AMD processors

Aweke et. al, ANVIL: software-based protection against next-generation rowhammer attacks.

So, how do you counted this attack? So, one of the very popular technique for preventing this is published in this work and essentially called as Anvil. The idea of this anvil is that you resides in the kennel and this basically is a kind of a good usage of performance counters which we have already seen how we can perform as I said we have we can use further performance counters as exploits.

But it can also be used for evaluations, it can be used for protection against side channels or side channel attacks. So, for example performance counters have got are available or intel sandy bridge and later microarchitectures which can be utilized.

(Refer Slide Time: 18:13)


ANVIL's Detection Process

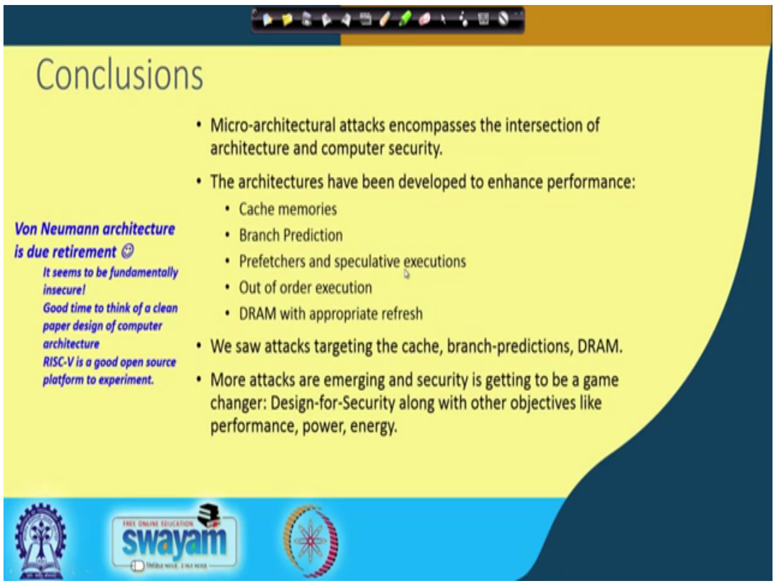
Monitor LLC Miss Rate
High Miss Rate?
No
Yes
Sample LLC Miss Addresses
Determine DRAM Rows of Samples
High Row Locality?
No
Yes
Determine Potential Victim DRAM Rows
Selectively Refresh Potential Victim DRAM Rows

- **Phase 1**: Monitor LLC Miss Rate
  - ❖ Is miss rate high enough ?
  - ❖ **Solution**: Monitor LLC miss count
- **Phase 2**: Sample LLC Miss Addresses
  - ❖ Do misses have high row and bank locality?
  - ❖ **Solution**: Sample misses out of LLC
- **Phase 3**: Remedy
  - ❖ How to protect potential victim rows?
  - ❖ **Solution**: Selectively refresh rows

So, the idea behind this strategy is as I illustrated here. So, you basically have you monitor the LLC miss rate and try to see that if the miss rate is high enough ok. So, you basically monitor the last level cache miss count and then you basically sample the LLC miss addresses and see that whether the misses have got high row and bank locality ok.

So, you can do that, you can sample the misses out of your LLC and then right what you do is you basically; the moment you basically have the suspect of this you kind of selectively refresh the rows and that would kind of thought the basic idea behind row hammers because the row hammers basically are the row hammer (Refer Time: 18:56) because our access is faster than the refresh cycle. So, if we can refresh it, then it will not work then therefore we can protect against the attack.

(Refer Slide Time: 19:06)



Conclusions

Von Neumann architecture is due retirement ☺
- It seems to be fundamentally insecure!
- Good time to think of a clean paper design of computer architecture
- RISC-V is a good open source platform to experiment.

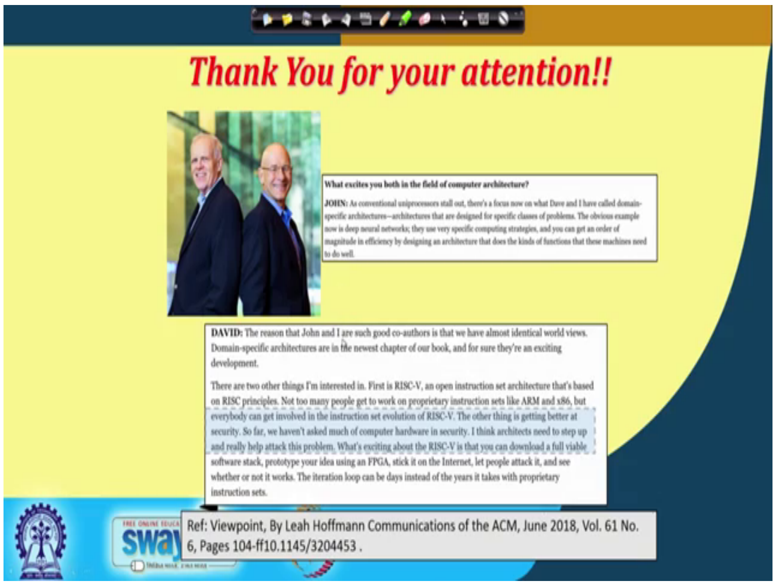
- Micro-architectural attacks encompasses the intersection of architecture and computer security.
- The architectures have been developed to enhance performance:
  - Cache memories
  - Branch Prediction
  - Prefetchers and speculative executions
  - Out of order execution
  - DRAM with appropriate refresh
- We saw attacks targeting the cache, branch-predictions, DRAM.
- More attacks are emerging and security is getting to be a game changer: Design-for-Security along with other objectives like performance, power, energy.

So, to conclude our discussions overall micro-architecture attacks encompass the intersection of architectures and computer security. So, we have seen several things and for example, we have seen the effect of cache memories, bunch predictions. We will not discuss in details about prefetchers and speculative executions, but this also has got a significant impact on the overall security. We have seen the effect of out of order execution for example, when we discussed about the effect of cache memories and then we discussed about DRAMs with appropriate refresh for example as an important design criteria.

So, we saw attacks targeting the cache, the branch predictions, the DRAM and more attacks are emerging as we are discussing and securities getting to be a game changer and therefore, design for security along with other objectives like performance, power, energy seems to be very important. And I would also like to make a comment yet that it seems like, the only one architecture is due retirement is to be, seems to be fundamentally insecure and it is a good time to think of a clean paper design of a computer architecture.

So, RISC-V is a good open source platform to experiment and we can actually potentially do several investigations and may be you know like kind of do investigations about various and try to think about a clean slide design of with security as an upfront design criteria.

(Refer Slide Time: 20:32)



Thank You for your attention!!

What excites you both in the field of computer architecture?

JOHN: As conventional uniprocessors stall out, there's a focus now on what Dave and I have called domain-specific architectures—architectures that are designed for specific classes of problems. The obvious example now is deep neural networks; they use very specific computing strategies, and you can get an order of magnitude in efficiency by designing an architecture that does the kinds of functions that these machines need to do well.

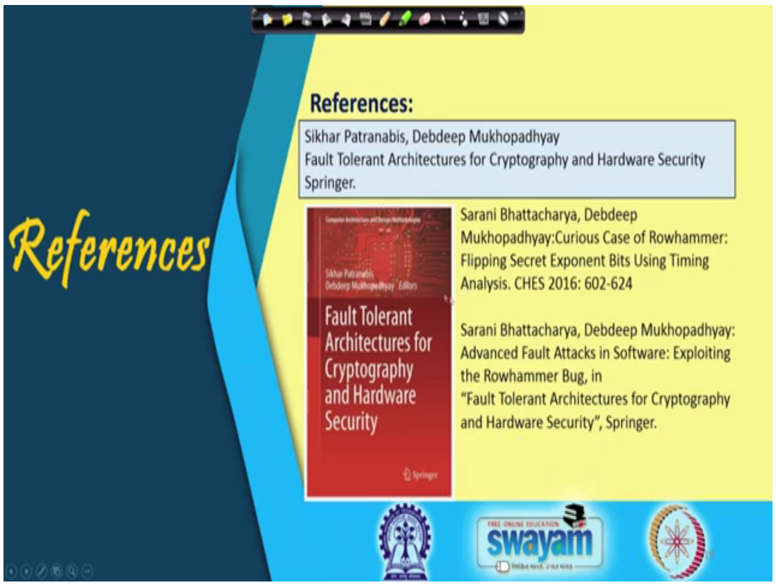
DAVID: The reason that John and I are such good co-authors is that we have almost identical world views. Domain-specific architectures are in the newest chapter of our book, and for sure they're an exciting development.

There are two other things I'm interested in. First is RISC-V, an open instruction set architecture that's based on RISC principles. Not too many people get to work on proprietary instruction sets like ARM and x86, but everybody can get involved in the instruction set evolution of RISC-V. The other thing is getting better at security. So far, we haven't asked much of computer hardware in security. I think architects need to step up and really help attack this problem. What's exciting about the RISC-V is that you can download a full viable software stack, prototype your idea using an FPGA, stick it on the Internet, let people attack it, and see whether or not it works. The iteration loop can be days instead of the years it takes with proprietary instruction sets.

Ref: Viewpoint, By Leah Hoffmann Communications of the ACM, June 2018, Vol. 61 No. 6, Pages 104-ff10.1145/3204453 .

So, just to quote you know like for example, the famous Hennessey Patterson in this context. So, in this discussion on view point published in communication of the ACM, there was an interesting comment which says that the other thing which we essentially are getting better at is or which we need to get better at the security. So, far we have asked much of computer hardware insecurity and I think that architects need to step up and really help attack this problem. And he says that what is exciting is the RISC-V is something which is essentially available open source and which we can potentially try to investigate to develop and implementation or processor architecture with security as an inbuilt design criteria.

(Refer Slide Time: 21:15)



References

References:

Sikhar Patranabis, Debdeep Mukhopadhyay
Fault Tolerant Architectures for Cryptography and Hardware Security
Springer.

Sarani Bhattacharya, Debdeep Mukhopadhyay:Curious Case of Rowhammer: Flipping Secret Exponent Bits Using Timing Analysis. CHES 2016: 602-624

Sarani Bhattacharya, Debdeep Mukhopadhyay: Advanced Fault Attacks in Software: Exploiting the Rowhammer Bug, in "Fault Tolerant Architectures for Cryptography and Hardware Security", Springer.

Fault Tolerant Architectures for Cryptography and Hardware Security
Sikhar Patranabis
Debdeep Mukhopadhyay Editors
Springer
swayam

So, with this is the reference that you can follow for our discussions on Rowhammers. So, this is the book which is published by Springer and you can get also an access to this work and several references which has been mentioned in this work. So, with this I would like to say thank you to you for your attention.

Thanks.