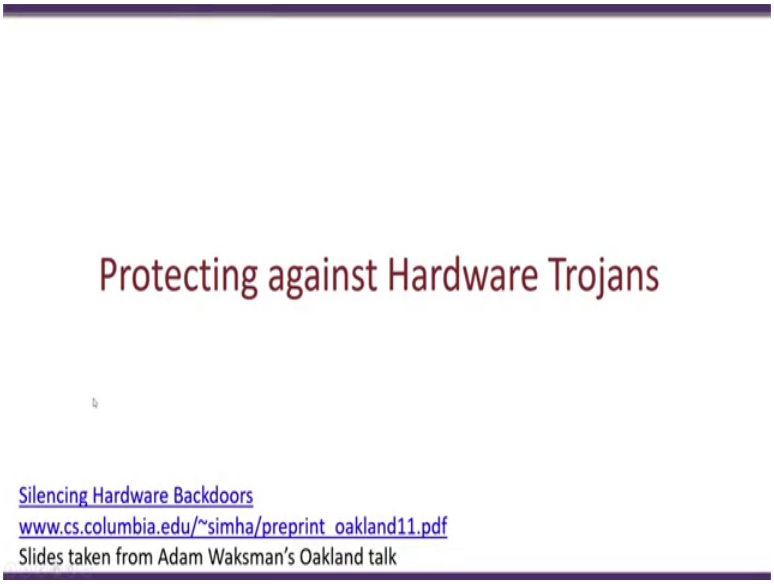
Information Security - 5 - Secure Systems Engineering
Professor Chester Rebeiro
Indian Institute of Technology, Madras
Protecting Against Hardware Trojans

Hello and welcome to this lecture in the course for Secure Systems Engineering, in the previous video lectures we had actually looked at hardware Trojans we had looked at two mechanisms to detect hardware Trojans, the first was called Fanci which detected hardware Trojans present in IP cores the second was using site channels such as the power consumption of a device to identify the presence of a hardware Trojan in a fabricated IC. Now as we mentioned in the previous video all of this techniques though successful to a certain extent are very easily awaited by Trojan developers essentially we could write hardware Trojans that specifically could bypass all of these techniques that have been developed to detect hardware Trojans.

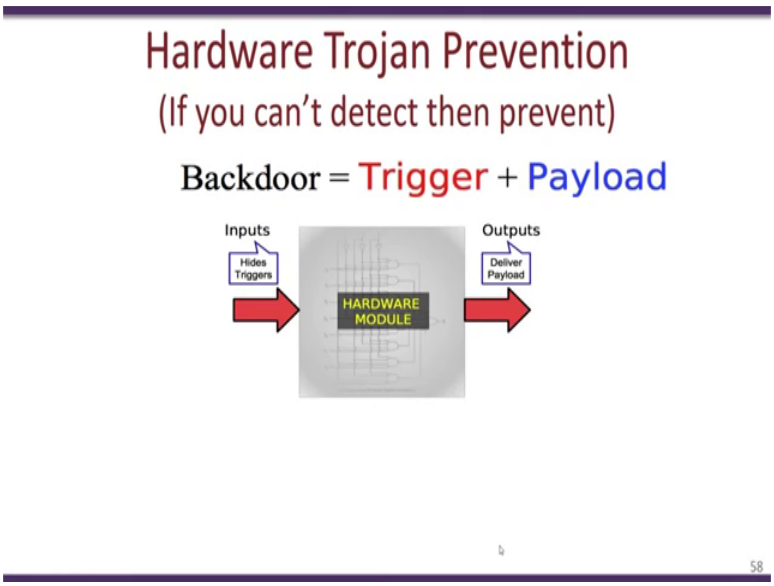
An alternate approach to solve this particular problem is to prevent hardware Trojans altogether so essentially if we cannot detect the presence of a hardware Trojan we will design circuits in such a way so that insertion of hardware Trojans in the circuits become difficult.

(Refer Slide Time: 01:30)



In this lecture we will be looking at this paper called silencing hardware backdoors this paper was actually presented in Oakland 2011 and a lot of the slides that we will be presenting today in this video lecture is by Adam Waksman's Oakland talk in 2011 essentially the talk corresponding to this specific paper.

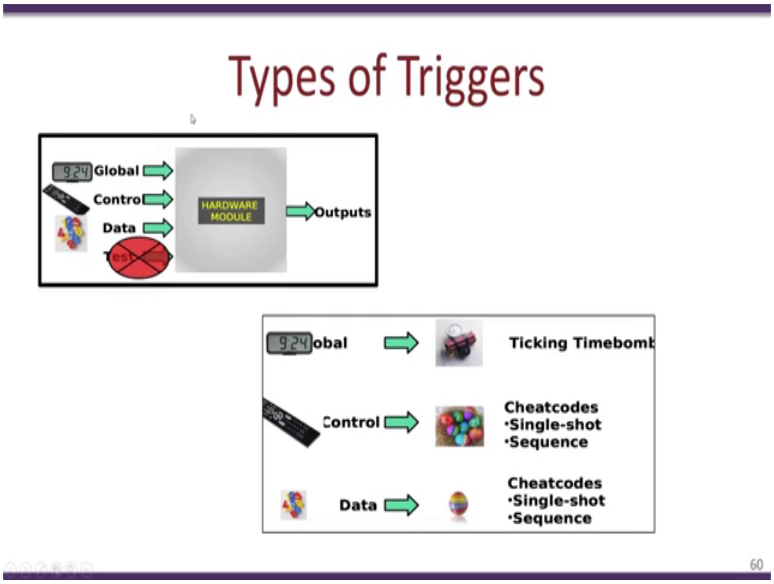
(Refer Slide Time: 01:52)



Now a lot of the paper is based on the fact that the hardware Trojan comprises of two components. The first component is known as a trigger and the second component as we have seen in the previous lectures is the payload if somehow we can design our hardware model such that it is difficult to create such a trigger then payload would never execute even though a hardware Trojan maybe present if the trigger never happens as expected by the attacker then the payload will never execute and a sensitive information or malicious activities by the device would never occur. So the critical aspect in what this paper talks about is how would we make triggering the hardware Trojan difficult? Essential idea in this entire thing is a way to hide the triggers so that the payloads never execute.

A base idea is to enumerate the different ways a triggers can appear in a hardware design and try to make it difficult for an attacker to actually design Trojans with these variety of triggers.

(Refer Slide Time: 03:07)

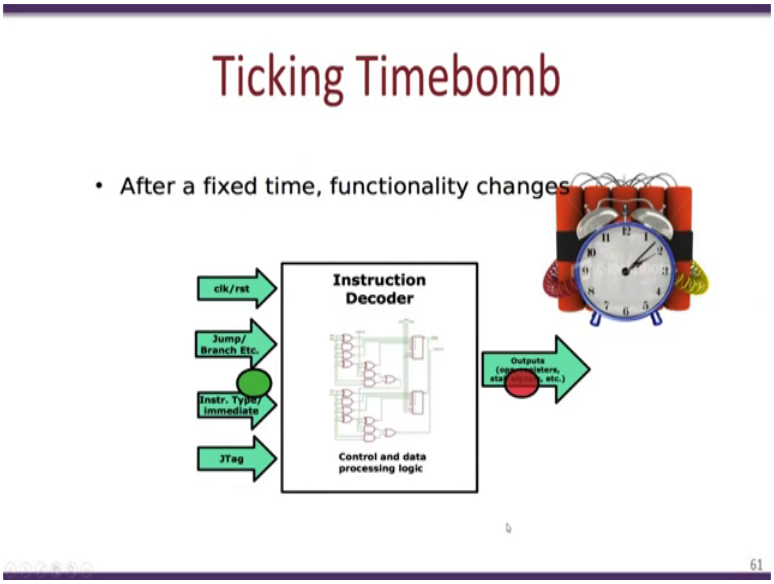


To understand this we take a high level picture of a hardware module and what we see is that every hardware module comprises of some inputs and (provided) provide some outputs these inputs can be categorized as global inputs such as the clock resets signal and so on. A control inputs which modify the state machine of the hardware module, data inputs for example data read from memory and so on and each of these will affect the outputs of the particular triggers and each of this will affect the outputs of this particular module, each of these inputs global control data and test could potentially be wave through which a trigger can be activated.

So in this particular work we focus mainly on the triggers that could come through the global signals the control signals and the data signals. So each of this different signals global control and data would have different types of triggers so we call the triggers based on the global signals such as a clock as a ticking time bomb, control signals could be cheat codes which are single shot or in a sequence or data signals could again be cheat codes in a single shot or sequence.

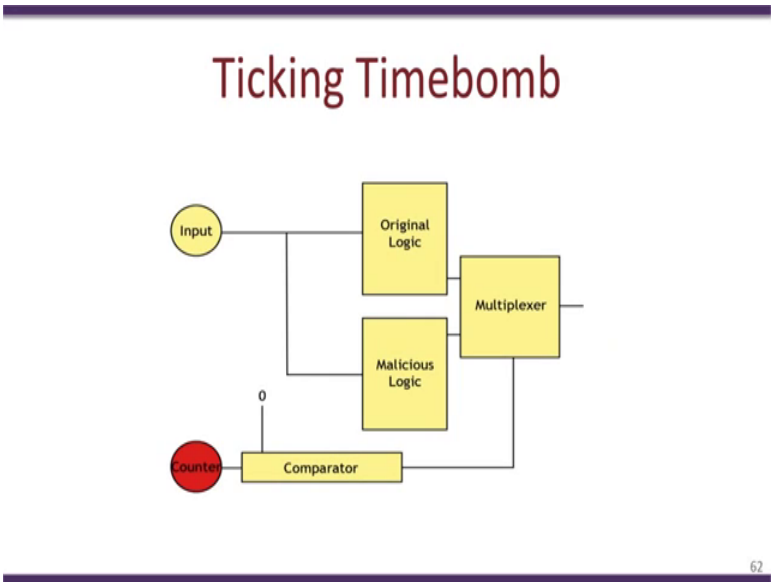
So what we will see in this particular video lecture is how each of this triggers will look and how we can design our hardware modules so as to prevent these triggers or make it difficult for an attacker to build hardware Trojans with this mechanisms.

(Refer Slide Time: 04:57)



So let us start with trigger ticking time bombs. Essentially if we consider this as our hardware module what a ticking time bomb trigger does is that it waits for a fix time and then triggers the hardware Trojans payload to execute this time is typically specified by the attacker. The attacker we want that the Trojan's trigger gets activated may say after a few months a few weeks or maybe even a few years or maybe at a specific time during the year.

(Refer Slide Time: 05:32)



So if we look at this more in detail about how a ticking time bomb's trigger would be designed the design would look something like this. so there would be an input to the device over here so

this input could be either say data or anything else which is specified by the user (there) this would go through an (original) through the standard logic which is supported by the hardware device and then the output goes. The hardware Trojan in such a scenario would look something like this. We would have a counter over here which possibly gets incremented at every clock pulse and furthermore this counter is compared with the triggered value.

For example let us say that the attacker has designed the hardware trigger so that it gets activated after a year. So this means that at regular intervals as the counter is incremented the comparator would check the number of clock cycles that elapsed or the number of amount of time that elapse and typically would give a value of zero after right amount of period has elapsed the comparator would provide an output of 1. Side by side there is a malicious logic which performs for example leaks the secret input through a multiplexer so in the general operation the comparator would give an output of zero which would mean that the original logics result is sent to the output.

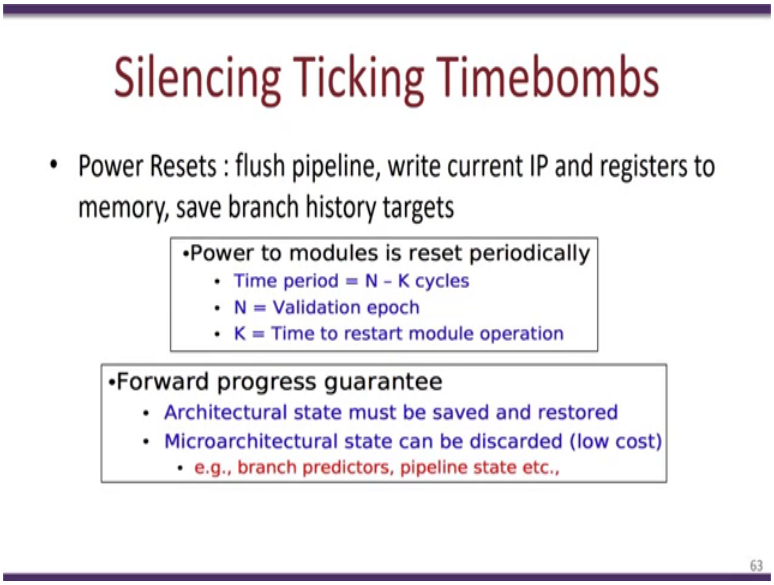
On the other hand when the right amount of time has elapsed the comparator would give you an output of 1 which causes the multiplexer to switch the output from that of the malicious logic, therefore when the comparator provides an output of 1 that is after the specified time has elapsed then the payload corresponding to this malicious logic gets executed and secret or sensitive data is leaked to the output. Now in order to silence ticking time bomb what we first assume is something known as an epoch duration. Now this duration could be anywhere say from 1 week, few days or even a month and what we assume here is that this during this epoch period we have extensively tested the entire design so that no hardware Trojans get triggered within this a particular epoch period.

Now beyond this epoch period we are not certain whether a hardware Trojan may get triggered or not. So in order to silence this ticking time bomb what we do is that at periodic intervals of time at periods equal to that of an epoch we reset the entire circuit that is we reset the power of the circuit. So by doing so what would happen is that the counter which is counting the time elapsed and is part of the hardware Trojan would also get reset. So therefore the maximum count that is permissible by this counter is upto the epoch. Now since we resetting the power at the end of the epoch the counter would reset and start counting gain to zero.

Now based on our assumption and the extensive testing that there are no hardware Trojan that can be triggered with a time less than an epoch resetting this circuit would prevent any Trojan

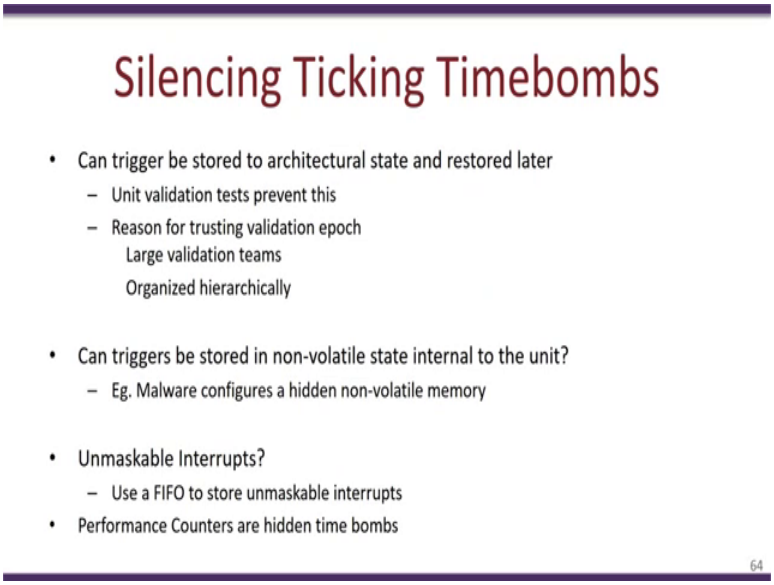
from being triggered. However resetting the circuit in the middle of its operation has its own hurdles.

(Refer Slide Time: 09:24)



So resetting would imply that we would need to flush the pipeline we need to save the current state of the entire system so that when the power is turned on again the hardware device can actually continue to execute from where it had stopped. The various components within the hardware such as register, branch history targets and so on would require to be saved.

(Refer Slide Time: 09:53)



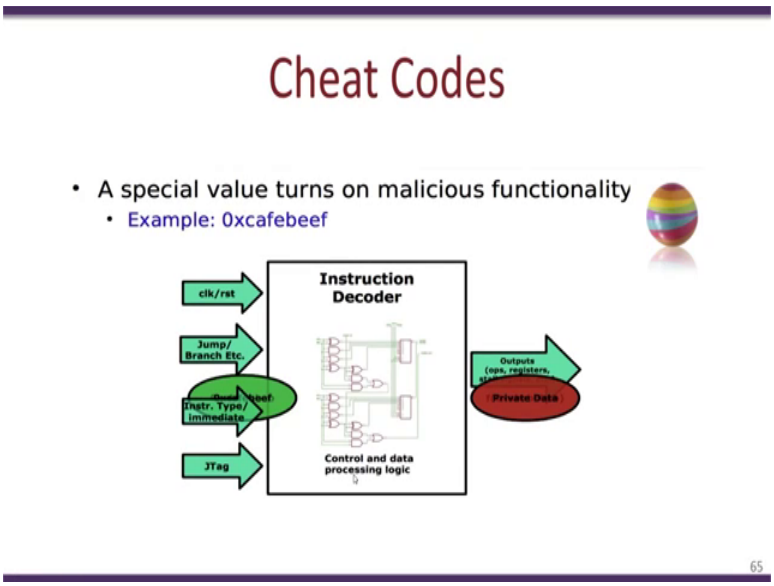
Now what we will now discuss is whether there is a way to bypass such protection mechanism, is there a way an attacker could still trigger Trojans payload at a time which is even greater than an epoch even with the resets that are present. So one thing that the attacker could do is that periodically the attacker could actually store the counter in a memory location. So this memory location we can assume is non-volatile and what we could actually do is that when the power is restarted the first thing the counter would so is to restart from the stored value this way potentially the attacker could cost the counter to count 2 value which is greater than the epoch time.

So in order to do this what the attacker would need is that some amount of non-volatile memory or flash memory to be present within the IC so the attacker could for example add a few cells of flash within the hardware design and use this malicious flash to store the intermediate counter values one thing that can be done to actually prevent this is to repeatedly turn the device on and off for example this can be done by say connecting the clock source to the power source therefore the device is continuously and very at a very high rate turned on and off and what would happen if there are flash memories present is that the flash memories would get destroyed.

Thus even if the attacker decides to add flash and store the intermediate counters in this non-volatile memory the flash would get destroyed by this repeated turning on and turning off the device. Another potential way to bypass this protection mechanism for a ticking time bomb is to write this intermediate value of the counter into some memory location in the RAM for instance and after the reset assuming that the RAM value has not decayed however aspect such as unmaskable interrupts could actually make designing such a mechanism quite difficult so in order as we know unmaskable interrupts cannot be blocked and if an unmaskable interrupt occurs during the time when the device is turned off or when the power reset is being done then that interrupt would get lost.

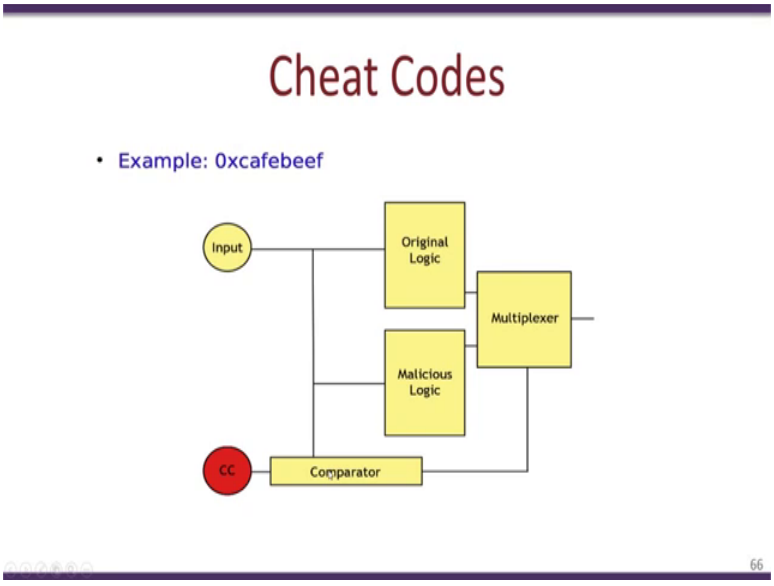
In order to prevent this what needs to be done is a FIFO can be used which would temporarily store the unmaskable interrupts during this power reset time. Other aspects which may make things difficult is the performance counters which may also be a source of time bombs.

(Refer Slide Time: 12:50)



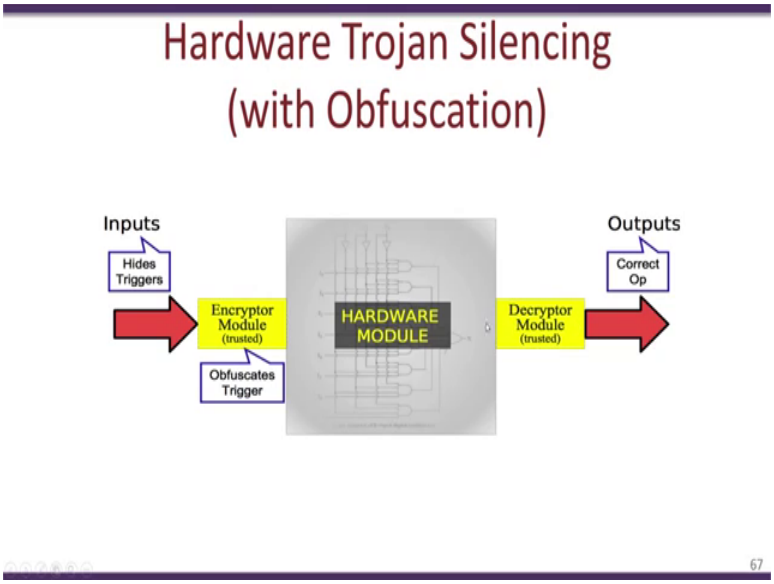
Another way to trigger Trojans is by something known as cheat codes so with a cheat code the specific input for example cafebeef when given to the trigger circuit would activate at the trigger circuit which in turn would then activate the payload of the hardware Trojan. Now we have seen examples of this in the previous videos we had seen how when a specific address is present in the input this specific address would then set a trigger value to 1 which would then switch a multiplexer to leak sensitive data to the output.

(Refer Slide Time: 13:29)



A typical cheat code type of hardware Trojan would look something like this we have a input over here and the attackers specific cheat code is stored over here. So for each input that comes there is a comparison done between the cheat code stored and a output of 0 or 1 is then obtained. So for inputs which are not equal to that of the cheat code the multiplexer would chose the original logic as the output and the results would work as normal and when the input is equal to that of the cheat code the multiplexer would then switch to the malicious logic and then the secret information get can get leaked out. So how would one actually prevent hardware Trojans from the activated with cheat codes.

(Refer Slide Time: 14:21)

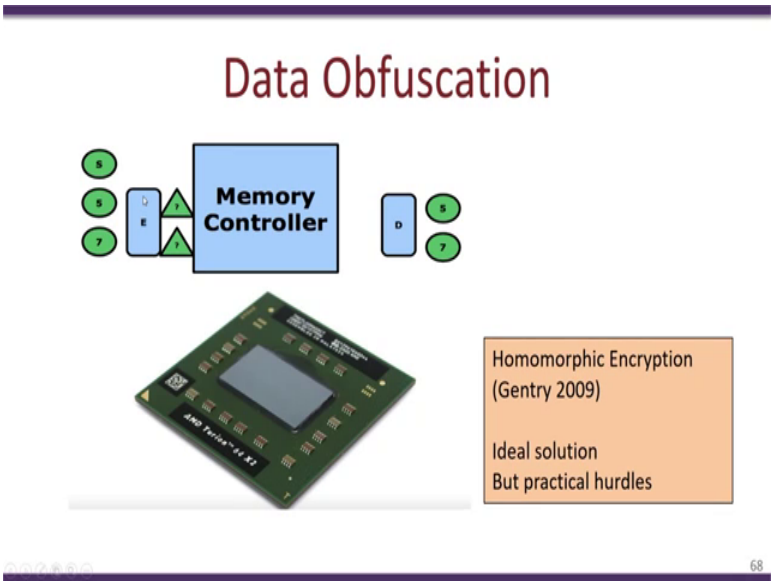


So one way is to make sure that the cheat code set by the attacker is never obtained as per his wishes. So for example if we have an encryptor module over here any input which is given first get encrypted or in other words gets obfuscates to some other value then the hardware module works on this encrypted data and then there is a decrypted module and obtained a current correct output. So note first that since or every encryptor there is also a decryptor at the output the results should always be correct and secondly also note that this obfuscation of the inputs would prevent the cheat code from activating the trigger.

For example if the cheat code is a cafebeef we have cafebeef stored over here now what the attacker would expect is that when he gives an input equal to cafebeef it would change the comparators output to 1 and forcing the malicious logic to be activated and change the output

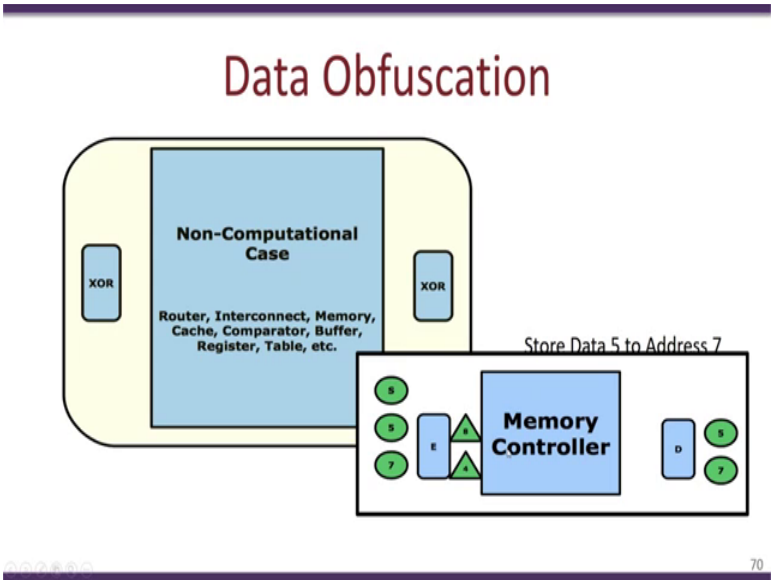
results. Now what we have done is we have introduce an encryptor over here which essentially obfuscates the input and cafebeef when supplied as an input is actually mapped to some other totally different value and therefore will not match the cheat code which is stored and therefore the attacker will not be able to control this multiplexer as per the wishes.

(Refer Slide Time: 15:56)



So one ideal way to build such data of obfuscation is by a technique known as homomorphic encryption this technique was actually proposed by (())(16:06) Gentry in 2009 so here what happens is that the memory controller is assumed to be able to work on encrypted data provide inputs for example S 5 and 7 it gets encrypted using the homomorphic encryption and then the memory controller will be able to function on this encrypted data and then provide its result. The decryptor would then be able to decrypt and get the correct values for the inputs. However homomorphic encryption is quite difficult to achieve in practice especially for this kind of uses and therefore we would require alternate options.

(Refer Slide Time: 16:55)



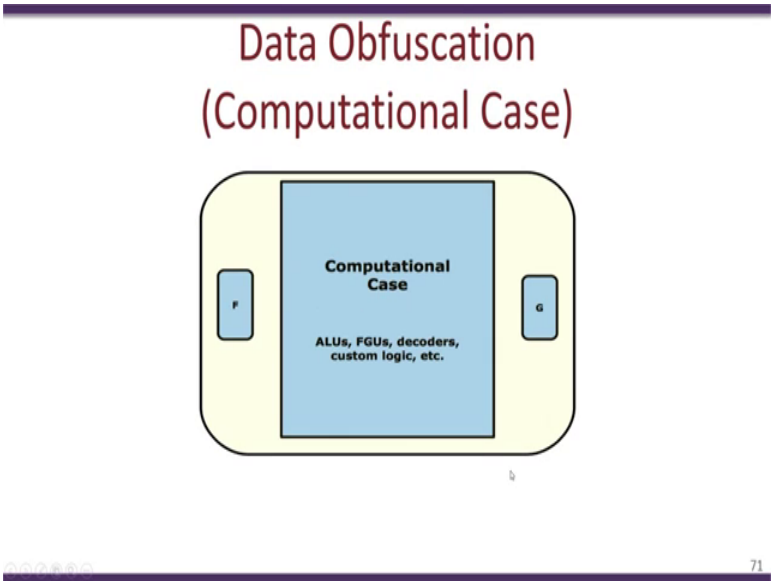
So what was proposed in the paper was to actually divide the type of operations into non-computational and computational. So hardware components such as routers, interconnects memory, cache memories, buffers, registers and so on are non-computational hardware entities. So these are non-computational because it does these do not actually modify the data but rather they just either store the data or just route the data to specific locations or just provide a look up so on. So providing data obfuscation with such non-computational hardware units is quite easy all that is required is just an EX-OR at the input with some secret data.

So this EX-OR would then map specific input to something which is totally different and similarly anything which is stored over here that same secret is EX-OR back to obtain the corresponding output. So for example if we have this memory controller what we could possibly do is if your input is 5 you EX-OR it with a certain specific key and obtain a value of 8. So a similarly by EX-ORING again the output 8 with that same key you re-obtain 5. In a similar way by taking the input 7 encrypting 8 with a specific key and obtaining 4 and at the other end when we have 4 we decrypt it with the same key to obtain back the 7.

So this would prevent the triggers now let us say that if in this memory controller there is a hardware Trojan which is based on this cheat code and let us assume that the cheat code has a value of let us say 5.now when the attacker sends a value of 5 hoping that the trigger would get

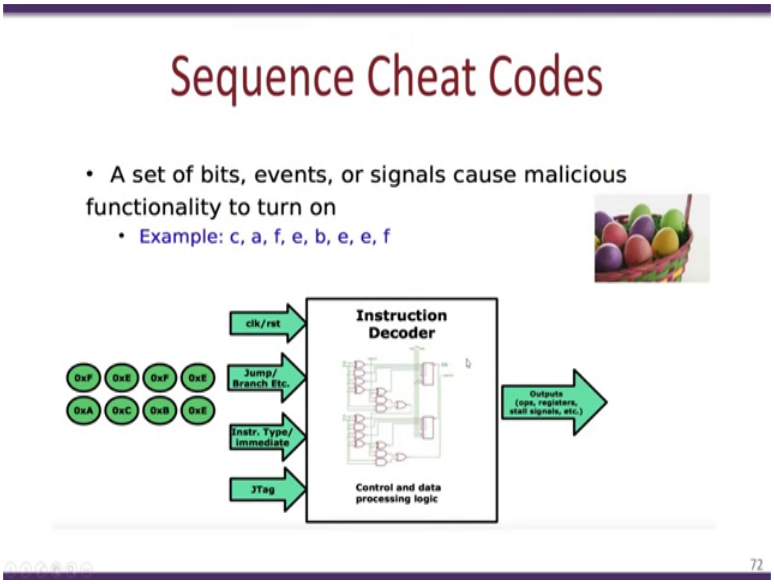
activated it will not in this case because infact the value of 5 is getting encrypted to 8 and therefore will not actually activate the trigger.

(Refer Slide Time: 19:06)



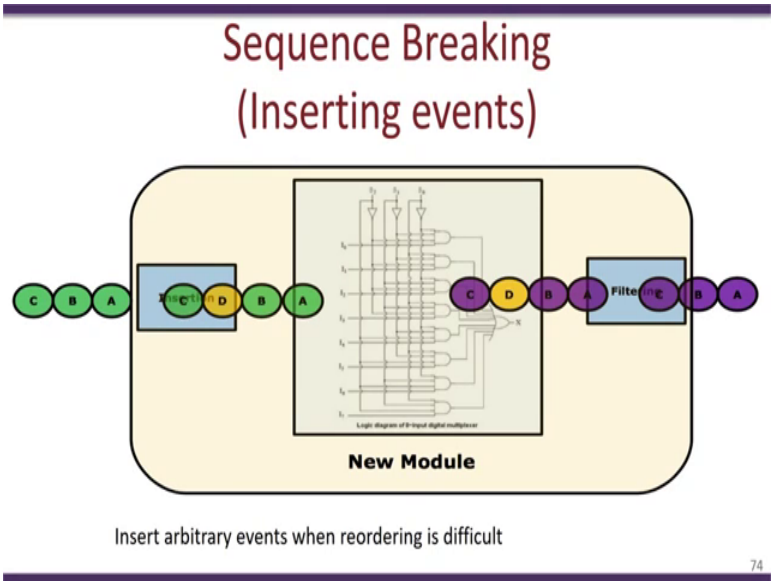
Now achieving data obfuscation for the computational case is far more difficult now if we have a module which actually computes something for example it would be ALU's (())(19:15) decoders or any other custom logic it is actually very difficult to obfuscate these things and this has to be done on a case to case spaces for example if there is an ALU and within the ALU there is let us say an a multiplier now we would require to obfuscate the inputs to the multiplier and remove the obfuscation at the after the n so that the original results are obtained. So this is not very easy and has to be done on our case to case spaces.

(Refer Slide Time: 19:49)



Another way triggers can be build is by sequence cheat codes for example if particular sequence such as CAFEBEEF is obtained in specific sequence only then this particular Trojan if it is present in this hardware unit it would get activated. So we had seen examples of this sequence cheat code in the first lecture on hardware Trojans and we had seen how a state machine would be maintain by the hardware Trojan to identify transitions whenever these specific inputs are obtained and eventually move the trigger from a 0 to a 1 and thereby by forcing the payload to be activated and secret information leaked out.

(Refer Slide Time: 20:40)

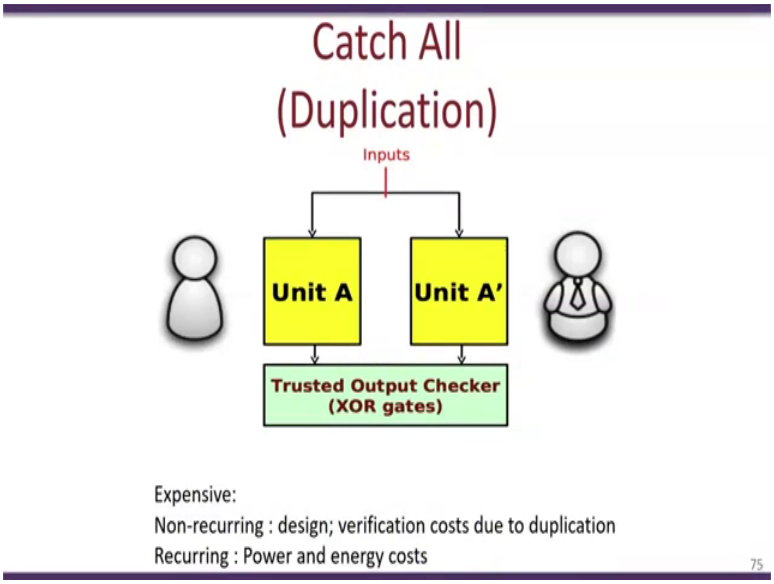


So one way by which we can break this sequence paste cheat codes is by reordering assuming that there is no dependency between these various inputs and any input in can independently come at any time if we break these inputs for example if the inputs provided in A, B and C order which is reordered the inputs like this into A C and B and feed it here and we reorder the results back after the computation is done. So why would this thing work? Let us say that the attacker has kept sequence cheat code as A B and C now the attacker is sending A B and C as the input and is hoping to hoping that this sequence of cheat codes would trigger the hardware Trojan and thereby the payload.

So internally this trigger will occur by the state machine which moves due to A then to B and then finally to C and moving to in this sequence would then cost the trigger to be activated. Now since we have reordered these inputs from A B C to A C B this state machine will never reach its final state and therefore the trigger will never get activated and therefore the payload will be non-functional. Another way to handle this sequence cheat codes is by (())(22:13) inserting some random inputs so for example over here given the input is A B and C and the attacker is actually waiting specifically for A to occur then B and then finally C in consecutive cycles inserting a random input D would actually break this sequence and (())(22:32) prevent the hardware Trojan from being activated.

So designing circuits or systems keeping in mind these hardware Trojans and the way a Trojans triggers can be designed could drastically reduce the cases where Trojans can be activated in particular hardware module however as we have seen there are many cases or many circuits where making such designs is incredibly difficult to do.

(Refer Slide Time: 23:04)



In such cases the worst case situation is where we could actually have two units possibly designed independently and both inputs are sent into both of these units and verified at the output. So for example we have a very complicated circuit over here let us say for example cryptographic algorithm and we want to ensure that this cryptographic algorithm does not have any Trojans.

What we could do is we could design completely independently and other unit a prime and possibly just fabricated this unit in a totally different environment and feed the same inputs to both this units. So both this units perform exactly the same functionality and if there is no Trojan that is present would give the same output. Now if for example we provide a specific input specific time or a specific cheat code which triggers a functionality in one of this units then the results would be different between unit A and unit A prime and as a result this checker over here would identify the difference and then stop the execution form occurring, thank you.