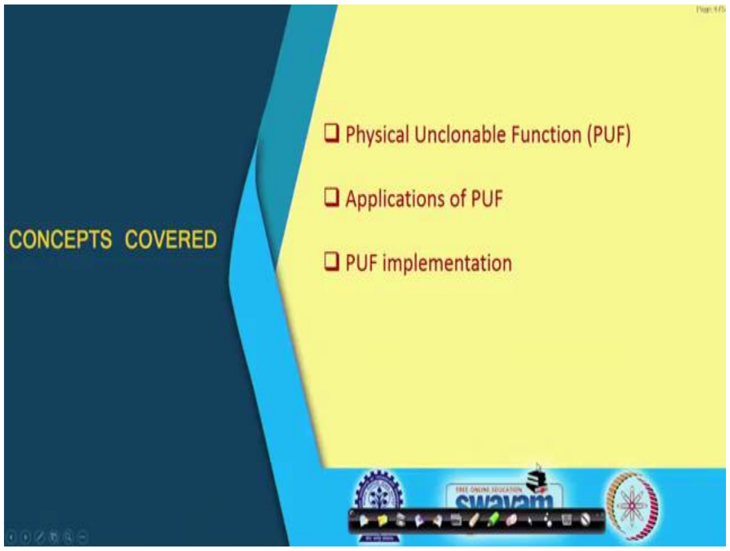
Ethical Hacking
Prof. Indranil Sengupta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 49
Physical Unclonable Functions

In this lecture, we shall be talking about Physical Unclonable Function or PUF. We mentioned earlier that this physical unclonable function plays a big role in designing systems in terms of hardwares such that it becomes secure. Means, in other words with respect to hardware security this physical unclonable function or PUF can be used to design secure systems in a relatively easier way. So, let us see what this PUF is actually all about.

(Refer Slide Time: 00:52)

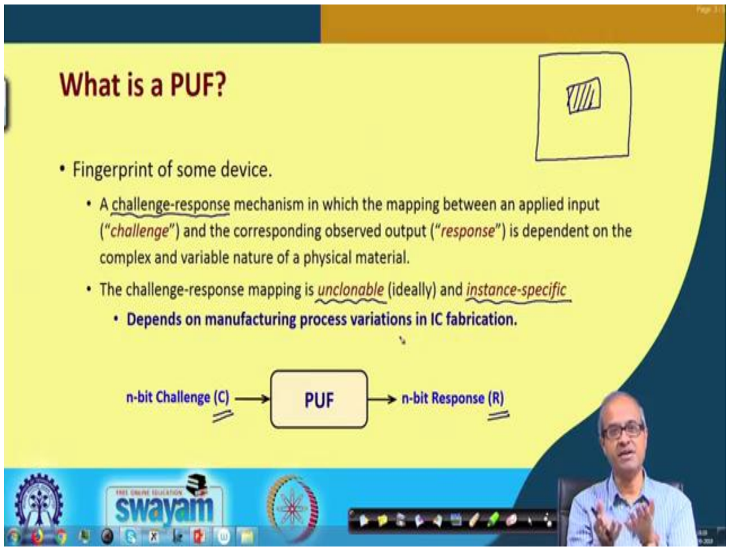


CONCEPTS COVERED

- Physical Unclonable Function (PUF)
- Applications of PUF
- PUF implementation

Now, in this lecture we shall first be talking about the basic concepts of physical unclonable function, then some applications of PUF and how they can be implemented ok.

(Refer Slide Time: 01:05)



What is a PUF?

- Fingerprint of some device.
  - A challenge-response mechanism in which the mapping between an applied input ("*challenge*") and the corresponding observed output ("*response*") is dependent on the complex and variable nature of a physical material.
  - The challenge-response mapping is *unclonable* (ideally) and *instance-specific*
    - **Depends on manufacturing process variations in IC fabrication.**

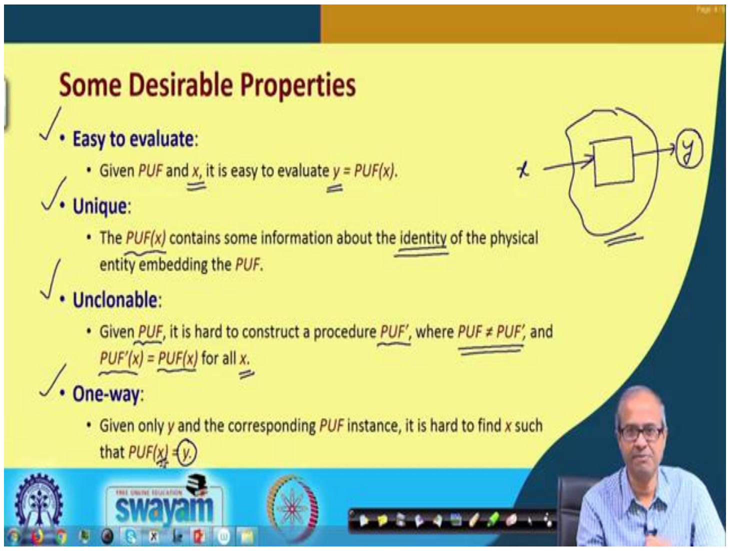
Talking about what is a physical unclonable function; you can say it is some kind of a fingerprint of some device. Now, how do you define a fingerprint? Suppose, as a human being my fingerprint is something which is supposed to identify me; my fingerprint is supposed to be unique. So, in the same way whenever I design some hardware circuit and IC chip, the idea is there must be something which should be unique to that IC chip. It can be acting as a some kind of a fingerprint of that particular device. So, PUF is something like that; the concept is that. It is defined as the fingerprint of some kind of a device.

Suppose, I have this kind of a PUF and we define something called challenge response mechanism. The way a PUF box is that suppose I can have a chip; I can have a chip and inside my chip the PUF can be sitting in between; this can be my PUF ok. Now, the idea is that when we use a PUF, there is a concept of a challenge and response pair or challenge response mechanism that comes into the picture. The idea is, suppose, I feed a challenge C; this is some data I feed as input; I call it as a challenge and I get an output which I call it as a response R. This C and R is referred to as the challenge response pair ok.

So, this PUF defines a mapping between this challenge and this response and the idea is that this PUF is something for which the challenge response pair is unique. If I design another IC chip where may be the same kind of a PUF I am building, but for that PUF the challenge response properties will be different. So, challenge response will be something like a fingerprint of that particular device ok. So, this is something which you define as unclonable, it cannot be copied, it cannot be cloned and instance specific. It is specific to that particular chip.

The way it is implemented this depends on manufacturing process variations. When multiple chips are fabricated there will always be small variations here and there. No two things can be exactly identical ok. So, that process variations, device variations that is what is exploited in the design of this PUF ok.

(Refer Slide Time: 03:58)



Some Desirable Properties

- **Easy to evaluate:**
  - Given *PUF* and *x*, it is easy to evaluate *y = PUF(x)*.
- **Unique:**
  - The *PUF(x)* contains some information about the identity of the physical entity embedding the *PUF*.
- **Unclonable:**
  - Given *PUF*, it is hard to construct a procedure *PUF′*, where *PUF ≠ PUF′*, and *PUF′(x) = PUF(x)* for all *x*.
- **One-way:**
  - Given only *y* and the corresponding *PUF* instance, it is hard to find *x* such that *PUF(x) = y*.

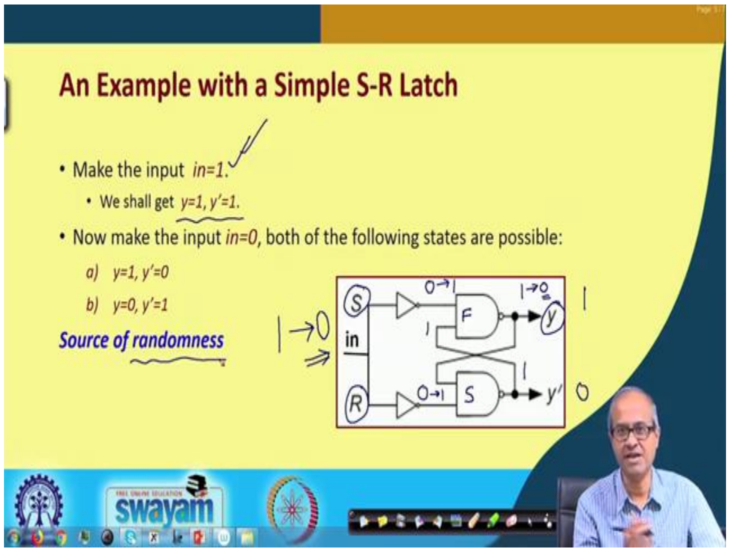
Some of the desirable properties of PUF are mentioned here. These are quite obvious. First is it should be something which should not be too difficult to evaluate in terms of the challenges response. Suppose, I give a challenge x, the response y should be easy to calculate, easy to compute ok. It should be easy to evaluate.

Secondly, for a particular device, the challenge response property should be unique. The value of PUF(x) for a particular x will contain some information about the identity of the, physical identity; that means, if I talk about a particular chip there is a PUF here, whatever x I feed here, the PUF of x if I call it y. So, whatever value I get y, this should uniquely identify this IC chip this should provide as the identity.

And, in the previous slide we have already mentioned there is a concept of unclonable; means given one PUF implement; that means, one chip, I have a PUF inside; if I have another copy of the same chip; there is also the same kind of a PUF, but the two PUFs will never be identical. One PUF let us call it PUF, other PUF call it as PUF'. They will always be unequal means for some given challenge, if you compute the response for the first one, compute the response for the second one, they will not be equal.

So, it is hard to construct two PUFs which are not the same such that for the same value of x their response will be the same; their response will be different that is the idea. And, it is a one way function that from y, from x you should get y, but given y you should not be able to get back x. It is quite similar to hash function calculations, the reverse mapping should be difficult ok.

(Refer Slide Time: 06:23)



An Example with a Simple S-R Latch

- Make the input *in=1*.
  - We shall get *y=1, y'=1*.
- Now make the input *in=0*, both of the following states are possible:
  - *a) y=1, y'=0*
  - *b) y=0, y'=1*

Source of randomness

Let us take a simple example of an S-R latch. I will just try to explain what this fabrication dependent or device dependent variations mean. Well, if you recall for those of you know how an S-R flip flop works; this is a single bit storage which store 1 bit of information, ok. There is one input “in” that same input is fed to this S and R let us say. This S and R are two different inputs; let us assume that the same input value “in” is fed to both S and R.

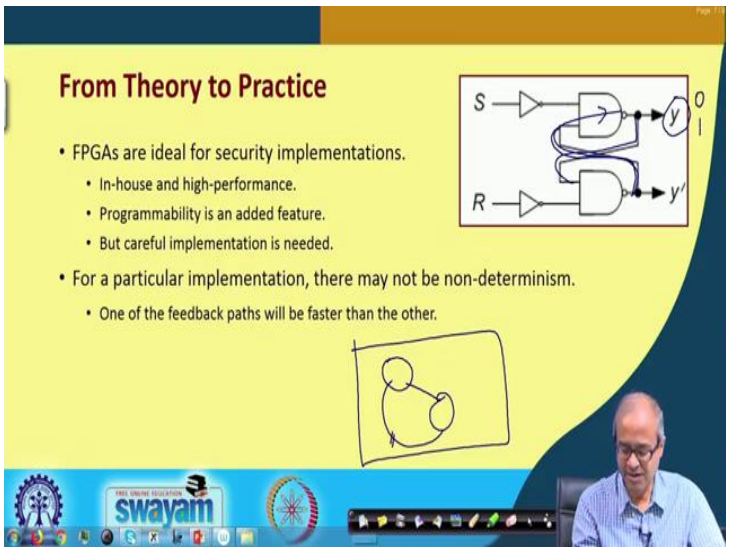
Now, let us say I apply 1 to input. If I apply a 1 here then the output of this NOT gates will be 0; this will be 0; these are NAND gates. So, one input 0 means the outputs will be

1; both the outputs will be 1; both y and y prime will be 1 ok. Now, let us say this in was 1; now I make it 0; I change it to 0. So, at this point in time both y and y prime were 1 and 1.

So, as soon as I make it 0 this outputs of the NOT gates will both become 1, this will become 1; this will also become 1. Now, here something happens; you see these are two gates. Now, two gates are fabricated in the chip. The two gates can never be exactly identical. Suppose, this gate is a little faster; let us call it F; this gate is a little slower; let us call it S. This gate faster means when this is 1, so both the inputs are 1 and 1, 1 and 1 NAND output is 0. So, this output will be changing to 0 first, because this gate is faster. This gate is slower, so, this output is still not 0. So, this will become 0 first and this 0 will be fed back; 0 and 1 this will remain as 1; but if it were the other way round, then this would have become 0; this would have become 1.

So, this output y whether it will finally become 1 or 0, it depends on the relative delays of the two gates which you cannot predict beforehand. So, here there is a source of randomness; it depends on fabrication ok. This is the basic idea behind which this kind of PUF design we are trying to build, fine.

(Refer Slide Time: 09:06)



From Theory to Practice

- FPGAs are ideal for security implementations.
  - In-house and high-performance.
  - Programmability is an added feature.
  - But careful implementation is needed.
- For a particular implementation, there may not be non-determinism.
  - One of the feedback paths will be faster than the other.

Now, from theory to practice usually these PUFs today the most of the research papers if you see, people have tried to build this PUFs around field programmable gate arrays or

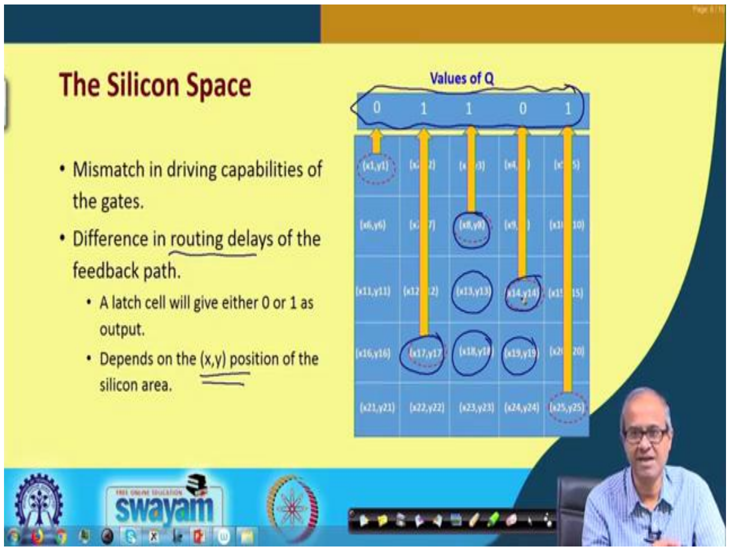
FPGAs. Many security protocols and implementations are based on FPGAs and people have also implemented PUFs inside FPGAs.

Now, the advantage of FPGAs is obvious; you can create a design and burn it on a FPGA in your lab, in-house. You can program it, you can change the design whenever you want. But, you should need to implement carefully when the implementation of a PUF is required, ok. See, for a particular implementation there may not be any non-determinism; like say you again look at that S-R flip flop, the same S and R; this kind of flip flop is there.

Suppose, in an FPGA implementation I design it in such a way that these two gates are placed in two different places in the chip; let us say one gate is placed here and one gate is placed here, and these two interconnecting lines, their lengths may not be same; one may be like this; one may be like this. So, one interconnection will be longer which means its delay will be longer. So, if this delay is longer, so that longer path can be the slower of the two; the other path will be faster.

So, it depends on the layout; the way you connect, place the two gates and connect them; that which gate effectively will be faster; which will be slower and whether the output finally in the example you took, whether it will be finally settling to 0 or to 1 right. So, this depends; you will have to understand that.

(Refer Slide Time: 11:09)



The Silicon Space

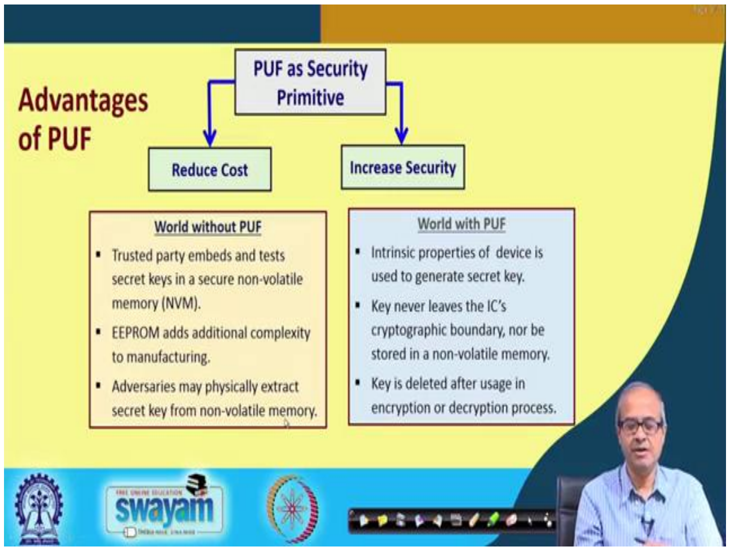
- Mismatch in driving capabilities of the gates.
- Difference in routing delays of the feedback path.
  - A latch cell will give either 0 or 1 as output.
  - Depends on the (x,y) position of the silicon area.

Now, if you look at a chip; here I am looking at the FPGA chip. So, these are the different grids. Let us assume that the NAND gate that I was talking about; that NAND gate I can potentially place in any of the grids; I can place it here; I can place it here; I can place it here; let us say I can place it here; I can place it here; I can place it here, so many places. Now, it depends where we are placing. Depending on that some of them if I set the input to 1 and then to 0, just in the previous example, some of them will be settling down to 0; some of them will be settling down to 1.

So, accordingly what will be the final values of y's that will depend on how you are interconnecting them? So, it does not depend on the circuit, but rather on how you are making the interconnection; where you are placing which gate. So, in FPGA particularly here I am not fabricating anything; you are mapping a design into a programmable fabric. So, where you are mapping, how are you interconnecting that will depend, that will determine what will be your final challenge/response pair of that PUF in terms of the S-R flip flop that we have just now talked about right.

So, the difference in the routing delays of these paths will determine that whether the flip flop output will be settling to 0 or 1. So, and that will depend on which location you are placing the flip flop, the x, y coordinates right.

(Refer Slide Time: 12:54)



Advantages of PUF

PUF as Security Primitive

Reduce Cost

Increase Security

World without PUF

- Trusted party embeds and tests secret keys in a secure non-volatile memory (NVM).
- EEPROM adds additional complexity to manufacturing.
- Adversaries may physically extract secret key from non-volatile memory.

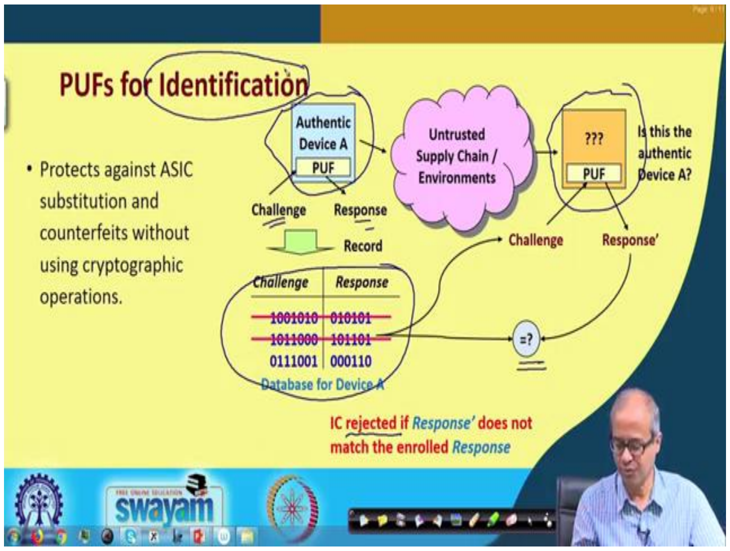
World with PUF

- Intrinsic properties of device is used to generate secret key.
- Key never leaves the IC's cryptographic boundary, nor be stored in a non-volatile memory.
- Key is deleted after usage in encryption or decryption process.

So, talking about the advantages of PUF, if you talk about PUF as a basic building block, you are using to design secure hardware, there are broadly two things – it reduces the cost of secure implementation; it also increases the security. Now, here I am showing two things. If there were no PUF, then trusted party embeds and tests secret keys in a secure non volatile memory; this is the typical thing which is done. Inside the chip some secret value is stored in a non-volatile memory that is used for encryption/decryption.

There is some kind of a flash memory, EEPROM and advisory on attacker can physically extract through side channel attacks. But, if you have a PUF, then some properties of the device that can be used to generate the secret key and the key will never be stored anywhere; the key will never come out of the IC chip. Depending on intrinsic property of some delays inside the chip, the key value will be automatically generated. And, the key is automatically deleted after is used. It is not stored anywhere. This is the advantage.

(Refer Slide Time: 14:18)



PUFs for Identification

- Protects against ASIC substitution and counterfeits without using cryptographic operations.

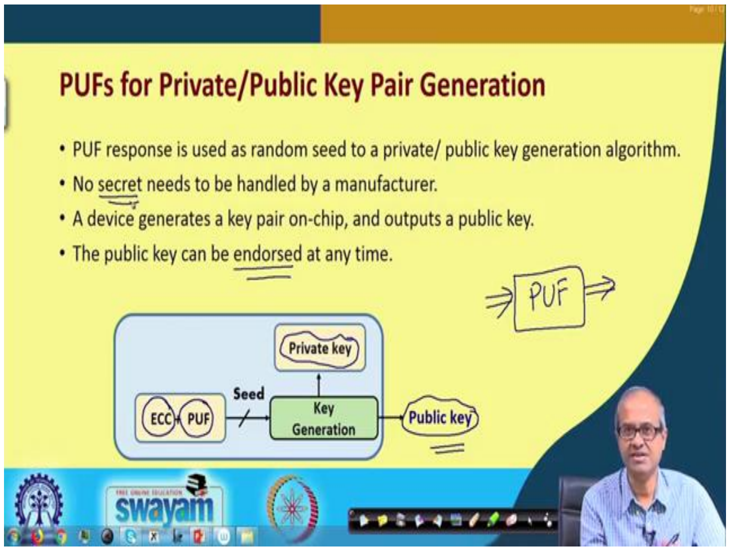
Challenge	Response
~~1001010~~	~~010101~~
~~1011000~~	~~101101~~
0111001	000110

IC rejected if *Response'* does not match the enrolled *Response*

There is another application of PUF where you want to identify; like I have fabricated or I have designed the device and someone has made a clone or a copy of my device; I want to find out which is the original and which is the copy. You see, for the authentic device which I had designed, I know what was my challenge/response properties. Let us say I have already created a table like that; this is the, for this challenge, this will be the response; for this challenge this will be the response.

Now, if some other device is given to me which contains a very similar PUF, I can consult an same table, I can apply this challenges and see what the responses are coming and I can compare whether they are equal or not, same or not and if they are not same, we can reject the IC, saying that this is not the same device. In this way you can identify a unique device, ok. Sometimes it is required; you need to identify a particular copy of an IC. You can use this kind of PUF IC fingerprinting for doing that.

(Refer Slide Time: 15:41)



PUFs for Private/Public Key Pair Generation

- PUF response is used as random seed to a private/ public key generation algorithm.
- No secret needs to be handled by a manufacturer.
- A device generates a key pair on-chip, and outputs a public key.
- The public key can be endorsed at any time.

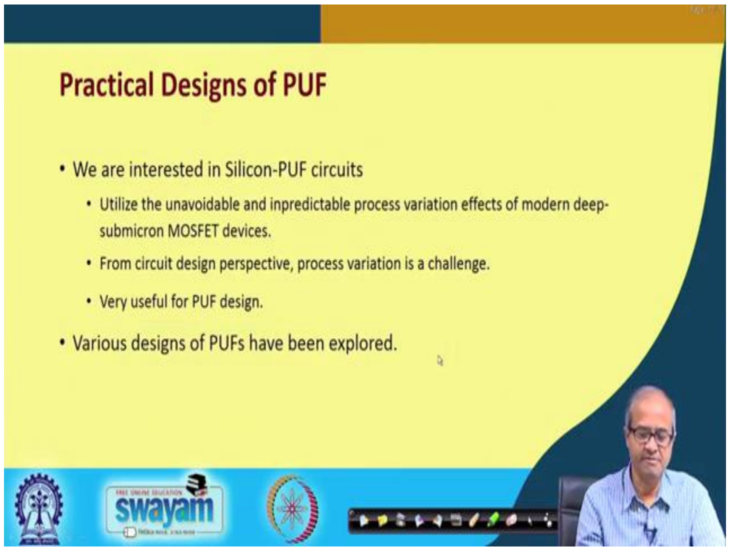
Now, there is another important application in public key cryptography. You can you PUF to generate public and private keys on chip. You need not have to rely on a trusted third party to generate the public/private key pair and deliver it to you through some mechanism; you can do it on chip.

The idea is like this; suppose, you have a PUF; you have a PUF. So, whenever you apply some input to a PUF, the response can be considered to be a random number. Response is random; it depends on the intrinsic properties of the device. It varies from one device to another. So, it is truly random. So, that random can be used as the initial value of the seed of a key generation and this PUF is typically used in conjunction with some public key algorithms; typically in hardware we use elliptic curve cryptography or ECC. So, ECC and PUF are sometimes used hand in hand. They are used together.

And, this with this ECC and PUF you generate a random seed through which the key generation model that can be inside the chip that can be generating the public key which can be distributed outside, but the private key will never leave the chip; it will remain inside. You are not storing anywhere and this private key can be generated online; you did not store it anywhere, ok. And, the public key also can be generated or endorsed anytime you want; you can again use that PUF to apply a particular challenge to get a response; the key generation module will be generating the same public key.

So, the manufacture need not store any secret key value inside the chip; that will automatically generated by the PUF mechanism, stored and generated by the chip itself right. So, this is how it works and in terms of cryptography, it helps a lot. It saves a lot of time and effort to manage the key, to secure the key inside the chip all right fine.

(Refer Slide Time: 18:14)



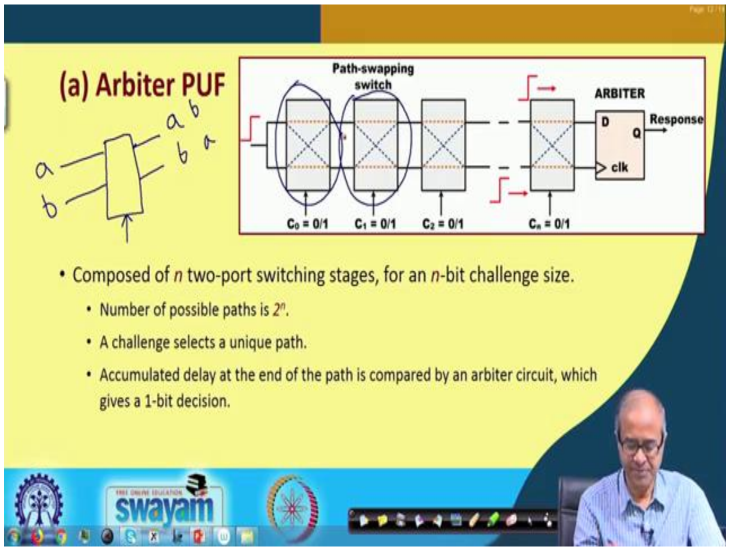
Practical Designs of PUF

- We are interested in Silicon-PUF circuits
  - Utilize the unavoidable and inpredictable process variation effects of modern deep-submicron MOSFET devices.
  - From circuit design perspective, process variation is a challenge.
  - Very useful for PUF design.
- Various designs of PUFs have been explored.

Now, in terms of the practical design, how the PUFs are actually implemented? Well, we are interested in chips right. So, we are interested in silicon PUF circuits which are implemented in silicon or CMOS. As I had said the basic idea is that when you fabricate chips, there will be process variations; from one chip to the other there will be some variations. So, those variations are exploited here.

But, with respect to design when circuit design is carried out, this process variation is a drawback. We try to make the variation as small as possible; but here the process variation is something which we are taking advantage of. We are trying to exploit the process variation and use it to our advantage right. So, this is quite useful for PUF design and a number of research papers and number of efforts are been carried out; various PUF designs have been explored. Some of them I am just briefly talking about.

(Refer Slide Time: 19:27)



(a) Arbiter PUF

- Composed of *n* two-port switching stages, for an *n*-bit challenge size.
  - Number of possible paths is 2^n.
  - A challenge selects a unique path.
  - Accumulated delay at the end of the path is compared by an arbiter circuit, which gives a 1-bit decision.

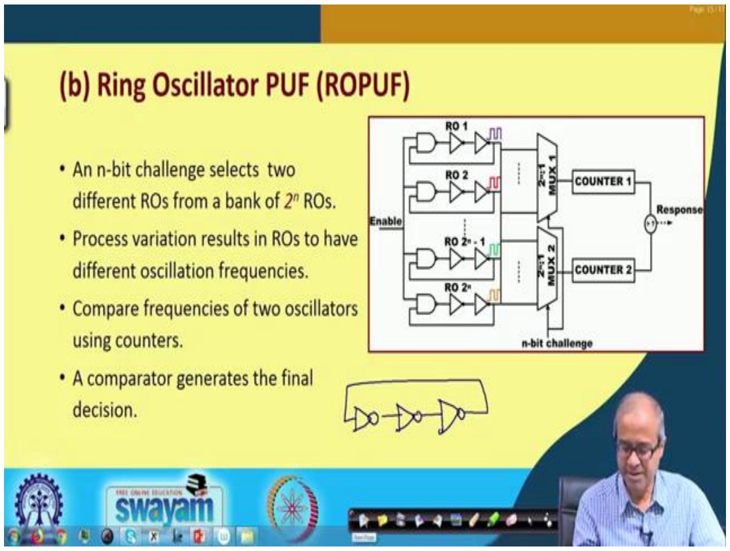
This is a very common and popular design. This is called arbiter PUF. Schematically, it is shown like this. You see these boxes you can see, rectangular boxes; these are nothing but path swapping switch. What it is actually is like this. Suppose, there are two inputs, let us say a and b. There are two outputs; either this input will be coming out as it is or they will be interchanged, a will come here; b will come here and what will happen that will be dependent on a control signal. So, depending on this control signal whether it is 0 or 1, either the inputs will be coming out straight or it will be exchanged right, ok.

Now, here what is happening; you see at the input we are applying a pulse; let us say 0 to 1 we are applying and this control signal combination, this is our challenge; we are applying some random bit pattern as a challenge, 0 1 1 0 1 0 1 1 or something. So, depending on that some random path is getting selected; some of them are going straight; some of them getting exchanged. So, the delays are different. So, when it reaches a final D flip flop the delay vary; that means, which inputs will reach first.

Here you see the same signal was fed to both of the inputs; but when it reaches here the two inputs maybe different; maybe little delayed with respect to each other; because they are following two different paths. The two paths are not identical, ok. And, the response we are storing in a D flip flop; one we are using as data; other we are using as clock. So, whatever will be storing in the flip flop that will be something unpredictable; it can be 0 or it can be 1. It depends on the relative delays of the flip flop; this is the basic principle behind arbiter PUF. There will not be one; there will be many such PUFs. So, the response will not be just one bit; there will be multi-bit response. There will be many such units placed in the chip in parallel; it will go on ok.

So, the same thing is mentioned here; there are n two-ports switching stages, for an n-bit challenge size. So, this is actually $n-1$, not n. This is 0 to C_{n-1}. So, number of possible path, for each of them there will be two possible paths, $2 \times 2 \times 2 \times 2$ ..., it will be 2^n. A particular challenge that we are feeding will be selecting a unique path and accumulated delay as I had said, is compared with the D flip flop which is the arbiter circuit which gives a 1-bit decision, 0 or 1. If there are multiple such arbiter circuits, it can give a k-bit decision, k-bit response.

(Refer Slide Time: 22:41)



(b) Ring Oscillator PUF (ROPUF)

- An n-bit challenge selects two different ROs from a bank of 2^n ROs.
- Process variation results in ROs to have different oscillation frequencies.
- Compare frequencies of two oscillators using counters.
- A comparator generates the final decision.

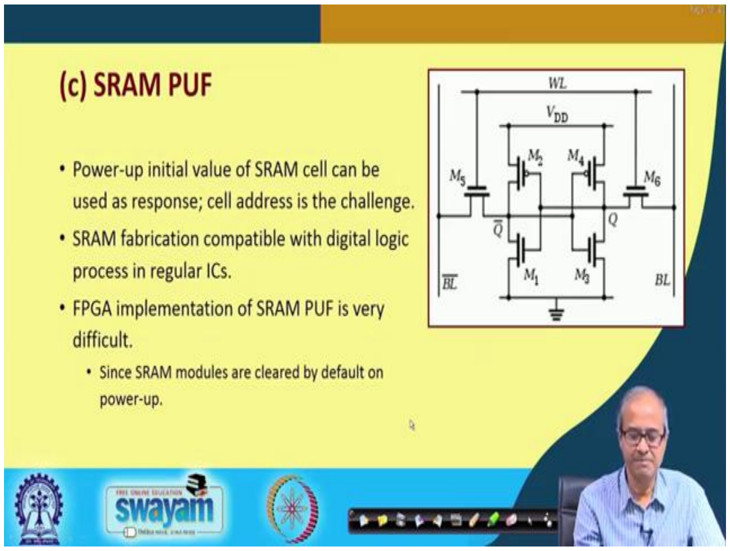
Now, there is another kind of PUF called ring oscillator PUF. So, what is a ring oscillator PUF? See; first let us understand what is a ring oscillator. If we take a inverter, the output I connect to the input, this acts as an oscillator. If I apply 0 output will be 1, 1 is feedback. Again, this 1 will become 0, 0 is feedback; again, this 0 will become 1, 0 1 0 1 0 1 this will go on. The same thing will happen if any odd number of inverters are connected like this, let us say 3 and I am connecting, same thing will happen; but the delay will be a little more; 0 1 0 1 0 1 like this it will go on oscillate ok. This is called ring oscillator.

Now, here the idea is that we have a number of ring oscillators you see. You ignore this gates in the first stage; you see is NOT gates are there and this NAND gate is also considered as a NOT gate. So, there are three inverter let us say. There are a number of such ring oscillators in the circuit right. So, the output is feedback here. Now, if it is enabled if the enable line is 1, if you feed a 1 here; say a NAND gate with one of the input 1 is equivalent to a NOT gate. So, if enable is 1, this becomes a NOT gate; it is a ring oscillator; if the enable is 0, then the output will be a steady one, it will stop oscillating right. This is how it works.

Now, the idea is that there are so many ring oscillators; they are all oscillating and oscillation means there will be some frequency of oscillation. So, what we do; using some challenge, we randomly select some ring oscillator. There will be several ring oscillator here; several ring oscillator here; let us say there are 2 to the power n. Just using an n bit challenge, we select two ring oscillators and their frequencies we measure in two counters. Then we compare whether the counter 1 is greater than counter 2 or not, which frequency is greater. This will be my final response, ok.

Now, depending on which ring oscillator you choose, these frequencies are different. So, the response will also get different; that means, whether the frequency will be greater or less. This is how randomness is generated ok. This is how ring oscillator PUF works.

(Refer Slide Time: 25:36)



(c) SRAM PUF

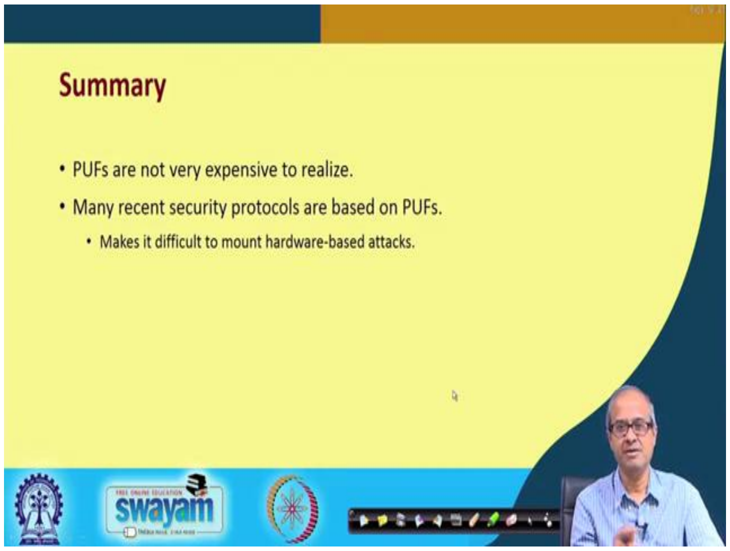
- Power-up initial value of SRAM cell can be used as response; cell address is the challenge.
- SRAM fabrication compatible with digital logic process in regular ICs.
- FPGA implementation of SRAM PUF is very difficult.
  - Since SRAM modules are cleared by default on power-up.

Then there is another kind of a PUF which is based on static random access memory, SRAM PUF. This is a typical diagram of a MOS based static random access memory cell, 1-bit storage. Now, the idea is that when you switch on power to a chip, suppose there is such a memory cell; initially the memory cell can start with an initial value of 0 or a 1; but you do not know what. It will be either 0 or 1 depending on the manufacture which transit is faster, which one will be switching first; that way if the output will be either set to 0 or 1.

So, if you have many such random access memory cells, depending on the manufacturing properties, they will be initialized to 0 or 1 randomly. This is the idea behind SRAM PUFs. The power-up initial values of SRAM cells are used as response and which cell you are selecting that is the cell address that will be the challenge. Randomly you select a cell, you see it is 0 or 1; randomly you select any another cell see it is 0 or 1 like that ok.

So, for a particular manufactured chip that should be the same; because for a particular property of this transistor device, devices; the way they are manufactured their delays, their gains, so when you switch on the power, it will be either 0 or 1; it will be deterministic; but across two chips it may be different ok. But, this SRAM PUF is more easy for an ASIC design; for a chip for an ASIC chip. But, for FPGA implementation SRAM PUF is very difficult because in FPGA whenever you are initializing the power all SRAM cells are reset to 0 by default; they are cleared. So, you cannot use this PUF using SRAM for FPGAs ok.

(Refer Slide Time: 27:46)



Summary

- PUFs are not very expensive to realize.
- Many recent security protocols are based on PUFs.
    - Makes it difficult to mount hardware-based attacks.

So, to summarize, this PUFs is something which is very simple; do not need much hardware to implement. Nowadays this circuit which I have shown, they are fairly simple, not much. So, they are relatively very inexpensive and using PUF you can make the security protocols that you are implementing in your chip much stronger; that is the basic idea and if you do that the kind of hardware based attacks we have been talking about it becomes much more difficult to mount this kind of attacks; your device becomes more secure ok.

So, with this we come to the end of this lecture, where we talked about PUF, physical unclonable function. In the next lecture, we shall be talking about Hardware Trojans which are also very much related to hardware security we mentioned earlier. We shall be seeing it in the next lecture.

Thank you.