Design for Internet of Things

Prof. T. V Prabhakar
Department of Electronic Systems Engineering
Indian Institute of Science, Bengaluru

Lecture - 07
Unique ID

So welcome back, today we will look at physically unclonable functions. And this is a topic which is I am sure it is very close to your heart because each one of us in India at least we have this Aadhar ID which is essentially collecting a lot of parameters from our biometric parameters and assigning an ID, a number to us. And this number is used for several transactions which involve us.

This could include administration of vaccines, the first dose and the second dose of vaccine, or it could be for know your customer with respect to certain investments, it could be for any operation that you want to do online which requires authentication of a person and ID of that person they use a ID and that ID is essentially the Aadhar ID. So quite like that if you are talking to billions of devices on the internet you actually want to know whether the data that you obtained actually came from that device.

It is very important. Let me give you an example. If you have, let us say, a part of an airplane and that part of an airplane has a sensor connected to it and that sensor is giving away the data. You actually want to know where that part was manufactured. You want to trace it back. So, traceability is important. Where was it manufactured? What was the original part? And was it manufactured at that point and what were the conditions under which that part was manufactured and whether that part now fitted into the airplane?

Several months later or several years later you still want to know whether it came from that original place, you want to trace it back to its origins. That is an important thing because you will then know that if there is an issue with respect to that sensor misbehaving or that particular part you want to know, understand a little more about the conditions under which that system was that sensor was manufactured or that part was manufactured.

And then you would actually know how many more such parts were there at that instant and it becomes easy for you to trace several things. So, traceability is a very important requirement. And in today's world where security is so prime in all what we do because 99% of us are actually well-behaved humans. 99.9% of us are actually well-behaved humans. It is that 0.1% that you need all the protection in order to ensure that we 99.9% live in peace.

So that being the small percentage of miscreants and kind of people who bring in malware and all of them that we have to have a huge, humongous amount of infrastructure to protect our devices and protect our infrastructure. So, security key generation is also an important thing in the same paradigm because every device will have to have an ID and that ID will actually correspond to a security key from which the data can be exchanged.

And you can do when you look at the security world you talk about a symmetric key, and you talk about a key which essentially is used for encrypting and the same key being used for decrypting. Or you could be talking of establishment of a session, on a session key. And then you can be talking about public and private key setup where you use a two key system where the session is established, and these two keys are used for one is for encrypting with the public key and then giving it to that end user.

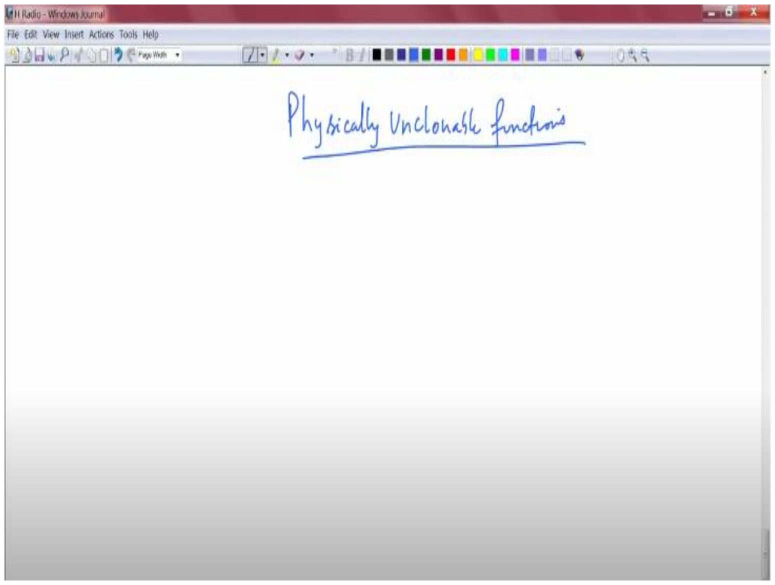
And the end user takes it and uses its private key and decrypts the data and so on. So, a lot of security protocols and security paradigms exist today. But we will not go into that detail. But the heart of it is that this key generation, source key generation actually depends largely on the ID of the device. Therefore, you must have this ID for the device for two reasons as we said. One is you want to go back to traceability you want to know from where it came.

And the second reason is you want to use it for generation of a security key. So, these two major reasons are required for sure when you are looking at requirement of a device ID. Conventional methods are one type. And what we should be talking about today is what are those; unconventional methods which are picking up? Which seem to hold a lot of promise particularly when you are talking of billions of devices. That brings us to another interesting topic.

And that topic is related to physically unclonable functions. The best way to understand this body of work is to read one of the latest papers which are written in that area. And I am going to go through that paper in detail for you. So, you get a feel for what PUF is all about. And then you will also be able to connect with a small experiment that we will try and show you that actually you can do this in your own labs, you can do this in your own homes with your Arduino devices and all that, that you may even have a small kind of a lab at home.

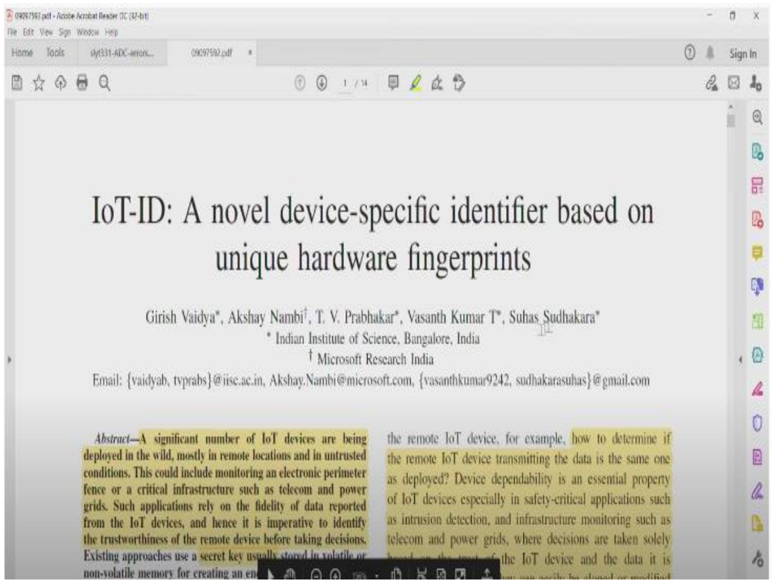
And you just be able to generate this from that device itself. So that is the exciting bubble that we have for you.

(Refer Slide Time: 06:25)



Let me point you to that paper which we wrote up a little a few months back.

(Refer Slide Time: 06:33)



IoT-ID: A novel device-specific identifier based on unique hardware fingerprints

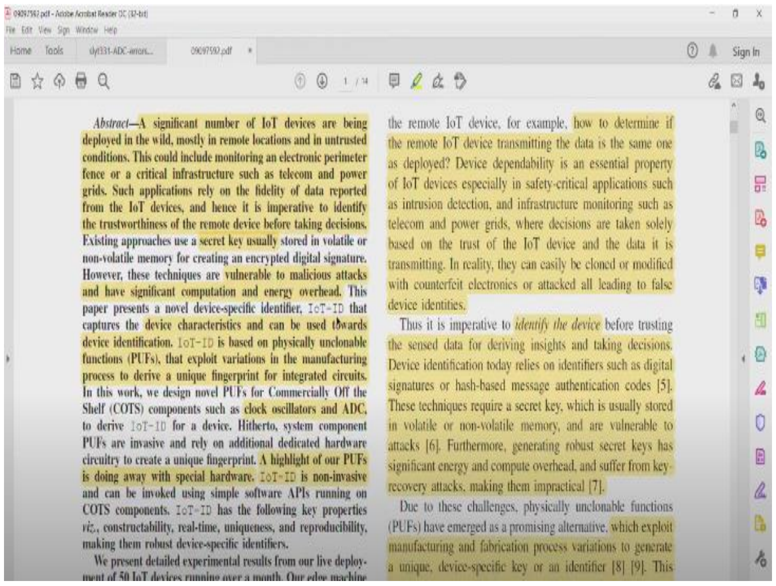
Girish Vaidya*, Akshay Nambi†, T. V. Prabhakar*, Vasanth Kumar T*, Suhas Sudhakara*
* Indian Institute of Science, Bangalore, India
† Microsoft Research India
Email: {vaidyab, tvprabs}@iisc.ac.in, Akshay.Nambi@microsoft.com, {vasanthkumar9242, sudhakarasuhas}@gmail.com

Abstract—A significant number of IoT devices are being deployed in the wild, mostly in remote locations and in untrusted conditions. This could include monitoring an electronic perimeter fence or a critical infrastructure such as telecom and power grids. Such applications rely on the fidelity of data reported from the IoT devices, and hence it is imperative to identify the trustworthiness of the remote device before taking decisions. Existing approaches use a secret key usually stored in volatile or non-volatile memory for creating an en

the remote IoT device, for example, how to determine if the remote IoT device transmitting the data is the same one as deployed? Device dependability is an essential property of IoT devices especially in safety-critical applications such as intrusion detection, and infrastructure monitoring such as telecom and power grids, where decisions are taken solely the IoT device and the data it is

And that paper is related to it is called the IoT-ID: A novel device-specific identifier based on unique hardware fingerprints.

(Refer Slide Time: 06:42)



Abstract—A significant number of IoT devices are being deployed in the wild, mostly in remote locations and in untrusted conditions. This could include monitoring an electronic perimeter fence or a critical infrastructure such as telecom and power grids. Such applications rely on the fidelity of data reported from the IoT devices, and hence it is imperative to identify the trustworthiness of the remote device before taking decisions. Existing approaches use a secret key usually stored in volatile or non-volatile memory for creating an encrypted digital signature. However, these techniques are vulnerable to malicious attacks and have significant computation and energy overhead. This paper presents a novel device-specific identifier, `IoT-ID` that captures the device characteristics and can be used towards device identification. `IoT-ID` is based on physically unclonable functions (PUFs), that exploit variations in the manufacturing process to derive a unique fingerprint for integrated circuits. In this work, we design novel PUFs for Commercially Off the Shelf (COTS) components such as clock oscillators and ADC, to derive `IoT-ID` for a device. Hitherto, system component PUFs are invasive and rely on additional dedicated hardware circuitry to create a unique fingerprint. A highlight of our PUFs is doing away with special hardware. `IoT-ID` is non-invasive and can be invoked using simple software APIs running on COTS components. `IoT-ID` has the following key properties *viz.*, constructability, real-time, uniqueness, and reproducibility, making them robust device-specific identifiers.

We present detailed experimental results from our live deployment of 50 IoT devices running over a month. Our edge machine

the remote IoT device, for example, how to determine if the remote IoT device transmitting the data is the same one as deployed? Device dependability is an essential property of IoT devices especially in safety-critical applications such as intrusion detection, and infrastructure monitoring such as telecom and power grids, where decisions are taken solely based on the trust of the IoT device and the data it is transmitting. In reality, they can easily be cloned or modified with counterfeit electronics or attacked all leading to false device identities.

Thus it is imperative to *identify the device* before trusting the sensed data for deriving insights and taking decisions. Device identification today relies on identifiers such as digital signatures or hash-based message authentication codes [5]. These techniques require a secret key, which is usually stored in volatile or non-volatile memory, and are vulnerable to attacks [6]. Furthermore, generating robust secret keys has significant energy and compute overhead, and suffer from key-recovery attacks, making them impractical [7].

Due to these challenges, physically unclonable functions (PUFs) have emerged as a promising alternative, which exploit manufacturing and fabrication process variations to generate a unique, device-specific key or an identifier [8] [9]. This

Now this paper if you read you will see that it talks about significant number of IoT devices are being used in the while. And you need to know that these sensors, sensing devices are used in critical infrastructure like telecom and power and so on. And also, I mentioned to you about the fact that you maintain a key using this device ID. But if you keep a key using that device ID in memory or you put it in the device it is vulnerable for malicious attacks.

People can simply take the device, connect the debugger, and then read the key. So, stealing the key is very simple if you are looking at storing a key in the memory. So, what is the key takeaway? You should not be able to store anything. If someone who puts the debugger and tries to look for a security key which is based on the device ID it should never be found. It should not be there.

Yet it should have an ID. So, you look what we are doing now. You do not carry an Aadhaar card in your pocket. You just go and then you use your biometric and you do a few things. And then it will quickly say that this must be the person. So, which means clearly auto identification is already happening. And if someone wants to verify it is very simple. You give the number, and they will pull out the signature, your fingerprint from it and then you put your fingerprint they both match. So, both ways it works. So that is the key point. I mean you are not storing anything.

It is actually a part of your body, a part of your system. It is the way you are wired. It is the way you are born. So that is something which is what you should look at even from the devices that we have. How close are they to the kind of IDs that we actually have? People use a lot of IDs. Do not forget that as kids we have seen these IDs cropping up in several ways. For example, people would identify a mole and say a mole on the left side of the chin is an indicator.

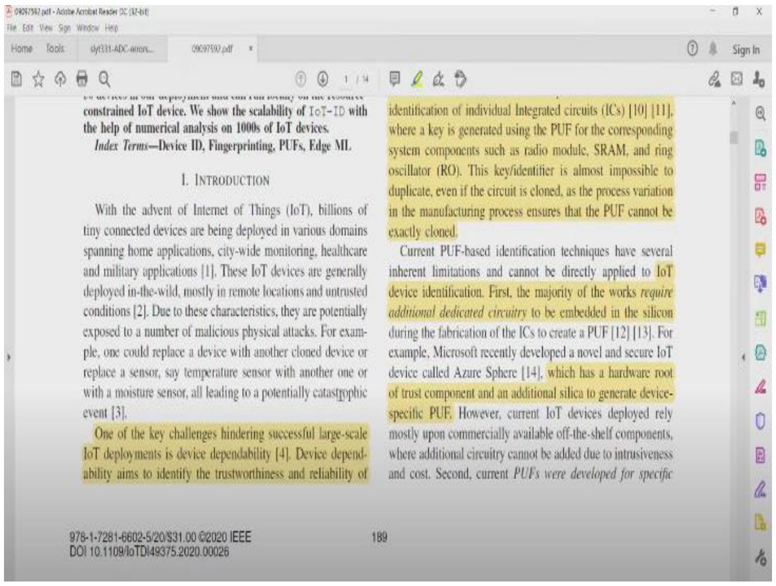
People would actually say what is your identification mark? Several decades ago, we used to use those things. It is not that people did not have that. So the mole is something that you cannot remove. So, it is part of your body. Say you see a mole on the left side then you know that this is so and so person is also identified by that. So that you can identify clones who look very similar to me but who are actually not me because eye can be deceptive.

If you look at the features, in fact, there is also a saying that there are seven duplicates of yourself in the world. So, I do not know whether it is true or not, but these things do exist. So, you need to identify that. So those seven duplicates that we are talking about may not actually have it, the mole on their left side of their cheek or whatever. So that has been the driving force behind how to do that kind of thing on devices.

Coming back here, you must be able to generate a physically unclonable function that exploits variations in manufacturing process to derive a unique fingerprint. Now this paper is very specific. It is talking about clock oscillators and ADC. Recall what I had discussed to you right in the beginning about a device, an IoT device. It has an oscillator circuit, it also has ADC and in fact, we said there are several ADC ports which are available connected to a MUX multiplexer.

And then actually there is an ADC which is actually converting the analog signal to digital signal. So now what we are saying is, can I use that ADC itself for the purpose of assigning a signature? Can I derive a unique signature from that ADC? Can I use the oscillator circuit to derive a signature for that ID? It turns out the answer is a big yes, you can. And that is the exciting part. So let us read this paper a little bit in detail.

(Refer Slide Time: 11:04)


09097592.pdf - Adobe Acrobat Reader DC (32-bit)
File Edit View Sign Window Help
Home Tools slyt331-ADC-errors... 09097592.pdf
Sign In

constrained IoT device. We show the scalability of IoT-ID with the help of numerical analysis on 1000s of IoT devices.

Index Terms—Device ID, Fingerprinting, PUFs, Edge ML.

I. INTRODUCTION

With the advent of Internet of Things (IoT), billions of tiny connected devices are being deployed in various domains spanning home applications, city-wide monitoring, healthcare and military applications [1]. These IoT devices are generally deployed in-the-wild, mostly in remote locations and untrusted conditions [2]. Due to these characteristics, they are potentially exposed to a number of malicious physical attacks. For example, one could replace a device with another cloned device or replace a sensor, say temperature sensor with another one or with a moisture sensor, all leading to a potentially catastrophic event [3].

One of the key challenges hindering successful large-scale IoT deployments is device dependability [4]. Device dependability aims to identify the trustworthiness and reliability of

identification of individual Integrated circuits (ICs) [10] [11], where a key is generated using the PUF for the corresponding system components such as radio module, SRAM, and ring oscillator (RO). This key/identifier is almost impossible to duplicate, even if the circuit is cloned, as the process variation in the manufacturing process ensures that the PUF cannot be exactly cloned.

Current PUF-based identification techniques have several inherent limitations and cannot be directly applied to IoT device identification. First, the majority of the works *require additional dedicated circuitry* to be embedded in the silicon during the fabrication of the ICs to create a PUF [12] [13]. For example, Microsoft recently developed a novel and secure IoT device called Azure Sphere [14], which has a hardware root of trust component and an additional silica to generate device-specific PUF. However, current IoT devices deployed rely mostly upon commercially available off-the-shelf components, where additional circuitry cannot be added due to intrusiveness and cost. Second, current *PUFs were developed for specific*

978-1-7281-6602-5/20/$31.00 ©2020 IEEE
DOI 10.1109/IoTDI49375.2020.00026 189

So, you will get a sort of a hang. One of the key challenges hindering large scale IoT deployments is device dependability. Device dependability aims at identifying the trustworthiness and reliability of the remote IoT device. We said this already. How to determine that the remote IoT device transmitting the data is the same one that was deployed? That is the point. I hope nobody has replaced that device.

And either by accident or by any from a vandalized perspective you do not want people to go around replacing devices. So particularly if it is a safety critical device you ought to be a lot more caution because if you are doing intrusion detection it may actually make a very selective decision that a particular type of intruder it may not even want to report. And things like that if there is a certain amount of power consumption which it is recording just the intruder may simply want the grid to collapse.

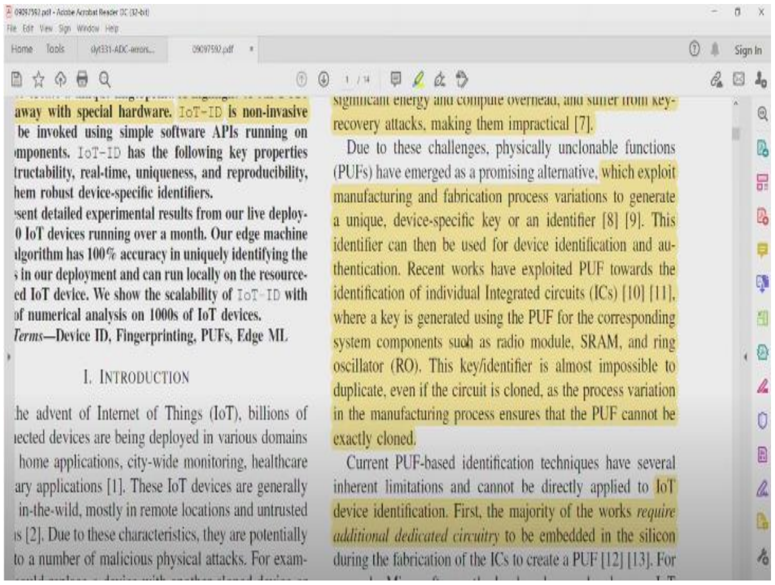
And may not even want to report a high current consumption or high-power consumption that has happened due to a spike you may not even want to report. So, such things can actually bring the grid down. So, such issues should not be you should think about it very carefully. There are what are known as IEDs in the power from the generation to before the actual distribution side, the supply side of the grid will have several IoT devices.

And there is something called an IED, intelligent electronic device that is actually you can look up. If you Google, you will find these are used in power grid systems. And if an IED device is not reporting data properly how will the national grid ever work? You should be very careful about such devices, and you do not want any malfunctioning IED devices. You do not want devices to be replaced.

For example, a service person goes there and finds that a particular IED is malfunctioning he or she removes that IED and puts a new IED and puts it back in the service room where it has to be serviced. Someone finds that and then goes and replaces it elsewhere. It is a disaster because it is a mistake. It is not that someone wanted to do it deliberately, but it was a mistake. But that can be costly.

So, such things you should be able to avoid. So, I hope the point that identifying the device is a very important thing before trusting the data from the sensor data. You want to ensure that it is trustworthy when you are actually using the data for any particular thing like things like that, I mentioned about IEDs and so on. Now what are these puffs?

(Refer Slide Time: 14:00)



away with special hardware. IoT-ID is non-invasive be invoked using simple software APIs running on mponents. IoT-ID has the following key properties tructability, real-time, uniqueness, and reproducibility, hem robust device-specific identifiers.
sent detailed experimental results from our live deploy- 0 IoT devices running over a month. Our edge machine lgorithm has 100% accuracy in uniquely identifying the s in our deployment and can run locally on the resource- ed IoT device. We show the scalability of IoT-ID with of numerical analysis on 1000s of IoT devices.
Terms—Device ID, Fingerprinting, PUFs, Edge ML

I. INTRODUCTION

he advent of Internet of Things (IoT), billions of nected devices are being deployed in various domains home applications, city-wide monitoring, healthcare ary applications [1]. These IoT devices are generally in-the-wild, mostly in remote locations and untrusted s [2]. Due to these characteristics, they are potentially to a number of malicious physical attacks. For exam-

significant energy and compute overhead, and suffer from key-recovery attacks, making them impractical [7].

Due to these challenges, physically unclonable functions (PUFs) have emerged as a promising alternative, which exploit manufacturing and fabrication process variations to generate a unique, device-specific key or an identifier [8] [9]. This identifier can then be used for device identification and authentication. Recent works have exploited PUF towards the identification of individual Integrated circuits (ICs) [10] [11], where a key is generated using the PUF for the corresponding system components such as radio module, SRAM, and ring oscillator (RO). This key/identifier is almost impossible to duplicate, even if the circuit is cloned, as the process variation in the manufacturing process ensures that the PUF cannot be exactly cloned.

Current PUF-based identification techniques have several inherent limitations and cannot be directly applied to IoT device identification. First, the majority of the works *require additional dedicated circuitry* to be embedded in the silicon during the fabrication of the ICs to create a PUF [12] [13]. For

09097592.pdf - Adobe Acrobat Reader DC (32-bit)
File Edit View Sign Window Help
Home Tools slyt331-ADC-errors... 09097592.pdf
Sign In
1 / 14

How can you actually do something equivalent of Aadhar ID from humans? Well, what they actually do is you exploit manufacturing and fabrication process variations to generate a unique device specific key or an identifier. This is the thing when it is wired when it is manufactured there are variations, process variations, like even humans you talk about twins. When you talk about twins there is some variation in the two twins.

It is not that they are what the eye looks very identical is actually not true. They are actually very different. Therefore, it is quite like that. You can have a lot of manufacturing and fabrication process variations. And you can exploit that to generate a device key. And this identifier can be used for device identification and authentication. So now recent works have exploited PUFs towards identification and individual integrated ICs where a key is generated using the PUF for the corresponding system components such as radio, SRAM, ring oscillator and so on.

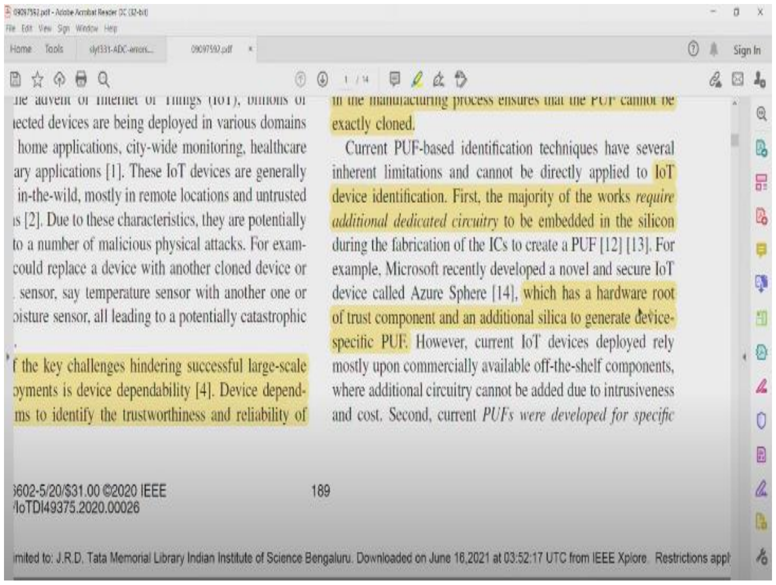
This key identifier is almost impossible to duplicate. It is almost impossible to duplicate even if the circuit is cloned, as the process variation in the manufacturing process ensures that the PUF cannot be exactly cloned. So that is the point. Now if you look at it is not that this area of PUF is something that has come out of the blue. It has been there for several years, more than a decade, if not more. The ring oscillator PUF is a very old concept.

And you can look up in the initial part itself people have been working on looking at how to generate these device IDs. So, it is been around. The point about this paper the one that we are reading together is you can do without any additional circuitry. With whatever is there you should be able to generate something very strong. Whereas ring oscillator PUF you have to wire a ring oscillator and put it there, then it will give you a unique ID.

So, this is something which is also catching up that we do non-invasive way of generating from commercially available components. You go to the market, you buy an IoT device, it could be let us say NodeMCU or a Raspberry Pi or a Nordic processor board or an Arduino board. You should be able to generate an ID quickly from it. And so, can I get a unique ID from that? So that is the point about this work.

It is all about how you can use existing components, existing oscillator, existing ADC. And can you generate anything from that? So that is the point.

(Refer Slide Time: 16:46)



he advent of Internet of Things (IoT), billions of
ected devices are being deployed in various domains
home applications, city-wide monitoring, healthcare
ary applications [1]. These IoT devices are generally
in-the-wild, mostly in remote locations and untrusted
s [2]. Due to these characteristics, they are potentially
to a number of malicious physical attacks. For exam-
could replace a device with another cloned device or
sensor, say temperature sensor with another one or
oisture sensor, all leading to a potentially catastrophic
.

f the key challenges hindering successful large-scale
oyments is device dependability [4]. Device depend-
ms to identify the trustworthiness and reliability of

in the manufacturing process ensures that the PUF cannot be exactly cloned.

Current PUF-based identification techniques have several inherent limitations and cannot be directly applied to IoT device identification. First, the majority of the works *require additional dedicated circuitry* to be embedded in the silicon during the fabrication of the ICs to create a PUF [12] [13]. For example, Microsoft recently developed a novel and secure IoT device called Azure Sphere [14], which has a hardware root of trust component and an additional silica to generate device-specific PUF. However, current IoT devices deployed rely mostly upon commercially available off-the-shelf components, where additional circuitry cannot be added due to intrusiveness and cost. Second, current *PUFs were developed for specific*

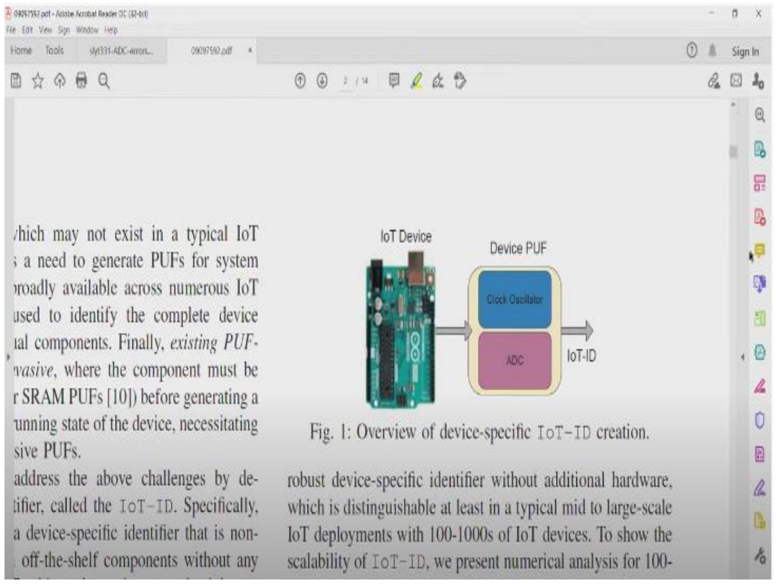
602-5/20/$31.00 ©2020 IEEE 189
/IoTDI49375.2020.00026

imited to: J.R.D. Tata Memorial Library Indian Institute of Science Bengaluru. Downloaded on June 16,2021 at 03:52:17 UTC from IEEE Xplore. Restrictions appl

So, this paper goes on to say that Microsoft also has been working quite intensively on this area and they have developed a secure IoT device called Azure Sphere which has a hardware root of trust component and an additional silica to generate device specific PUF. So again, there is something in additional component that is required, trust component basically. So let us put this down a little bit and then move on from here.

So, I think this paper you can read, but do not worry so much about it, if you got the idea that is already very good. Let us see what this paper actually does.

(Refer Slide Time: 17:30)



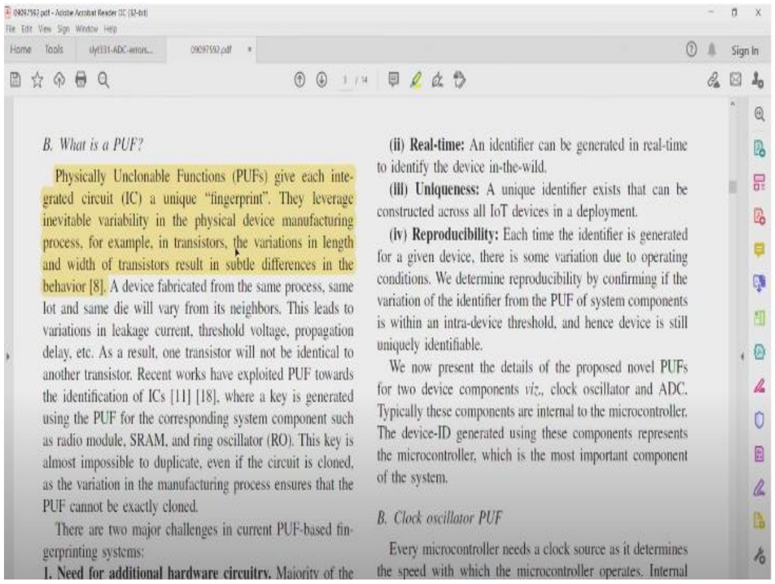
hich may not exist in a typical IoT
a need to generate PUFs for system
roadly available across numerous IoT
used to identify the complete device
al components. Finally, *existing PUF-*
vasive, where the component must be
r SRAM PUFs [10]) before generating a
unning state of the device, necessitating
sive PUFs.
address the above challenges by de-
tifier, called the IoT-ID. Specifically,
a device-specific identifier that is non-
off-the-shelf components without any

Fig. 1: Overview of device-specific IoT-ID creation.

robust device-specific identifier without additional hardware,
which is distinguishable at least in a typical mid to large-scale
IoT deployments with 100-1000s of IoT devices. To show the
scalability of IoT-ID, we present numerical analysis for 100-

Look, this is something you can connect very easily. You have an Arduino which is very familiar to you; can I generate a device PUF from this Arduino? This paper is just about that. You should be able to do it just like that in your labs and use it for whatever purpose that you want to. So, a clock oscillator and ADC these are the two things that you should be able to use. Now what is the key idea? How does this clock oscillator and ADC actually work? We will come to that. But before that formally define what a PUF is.

(Refer Slide Time: 18:02)


B. What is a PUF?

Physically Unclonable Functions (PUFs) give each integrated circuit (IC) a unique "fingerprint". They leverage inevitable variability in the physical device manufacturing process, for example, in transistors, the variations in length and width of transistors result in subtle differences in the behavior [8]. A device fabricated from the same process, same lot and same die will vary from its neighbors. This leads to variations in leakage current, threshold voltage, propagation delay, etc. As a result, one transistor will not be identical to another transistor. Recent works have exploited PUF towards the identification of ICs [11] [18], where a key is generated using the PUF for the corresponding system component such as radio module, SRAM, and ring oscillator (RO). This key is almost impossible to duplicate, even if the circuit is cloned, as the variation in the manufacturing process ensures that the PUF cannot be exactly cloned.

There are two major challenges in current PUF-based fingerprinting systems:

1. Need for additional hardware circuitry. Majority of the

(ii) Real-time: An identifier can be generated in real-time to identify the device in-the-wild.

(iii) Uniqueness: A unique identifier exists that can be constructed across all IoT devices in a deployment.

(iv) Reproducibility: Each time the identifier is generated for a given device, there is some variation due to operating conditions. We determine reproducibility by confirming if the variation of the identifier from the PUF of system components is within an intra-device threshold, and hence device is still uniquely identifiable.

We now present the details of the proposed novel PUFs for two device components *viz.*, clock oscillator and ADC. Typically these components are internal to the microcontroller. The device-ID generated using these components represents the microcontroller, which is the most important component of the system.

B. Clock oscillator PUF

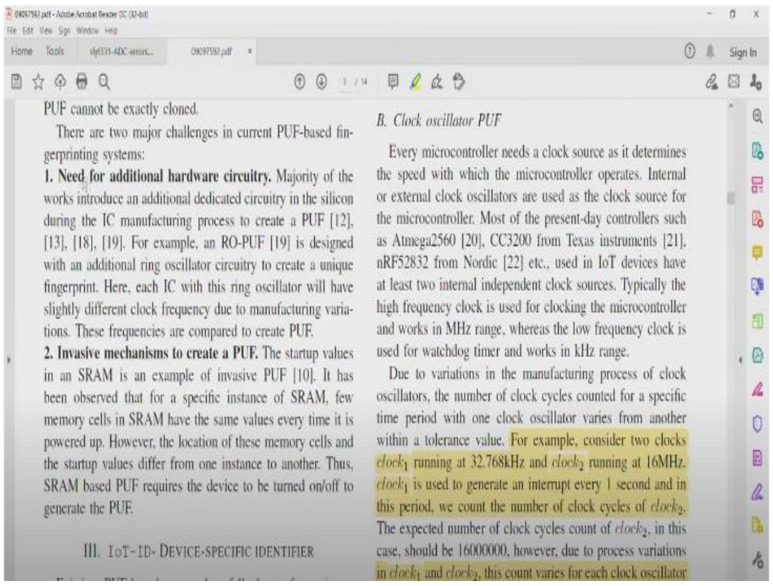
Every microcontroller needs a clock source as it determines the speed with which the microcontroller operates. Internal

PUF gives each integrated IC a unique fingerprint. This is clear. They leverage inevitable variability in the physical device manufacturing process. So anytime the chip is manufactured you will have within the same manufacturing process within the same die you can have variations. For example, in transistors the variation in length and width of the transistors result in subtle differences in the behaviour. See this is a strong statement.

A device fabricated from the same process, same lot, same die will vary from its neighbors. This leads to variations in leakage current, threshold voltage, propagation delay etcetera. So many signatures are already there. They are very different. From chip to chip they are different, leakage current is different, threshold voltage is different, propagation delay is different. And as a result, one transistor will not be identical to another transistor.

And people have been exploiting this to other works. I mentioned to you that one of the first things that came up was the ring oscillator PUF. It is also called the RO PUF. So that is been around for some time.

(Refer Slide Time: 19:25)


PUF cannot be exactly cloned.

There are two major challenges in current PUF-based fingerprinting systems:

1. Need for additional hardware circuitry. Majority of the works introduce an additional dedicated circuitry in the silicon during the IC manufacturing process to create a PUF [12], [13], [18], [19]. For example, an RO-PUF [19] is designed with an additional ring oscillator circuitry to create a unique fingerprint. Here, each IC with this ring oscillator will have slightly different clock frequency due to manufacturing variations. These frequencies are compared to create PUF.

2. Invasive mechanisms to create a PUF. The startup values in an SRAM is an example of invasive PUF [10]. It has been observed that for a specific instance of SRAM, few memory cells in SRAM have the same values every time it is powered up. However, the location of these memory cells and the startup values differ from one instance to another. Thus, SRAM based PUF requires the device to be turned on/off to generate the PUF.

III. IoT-ID- DEVICE-SPECIFIC IDENTIFIER

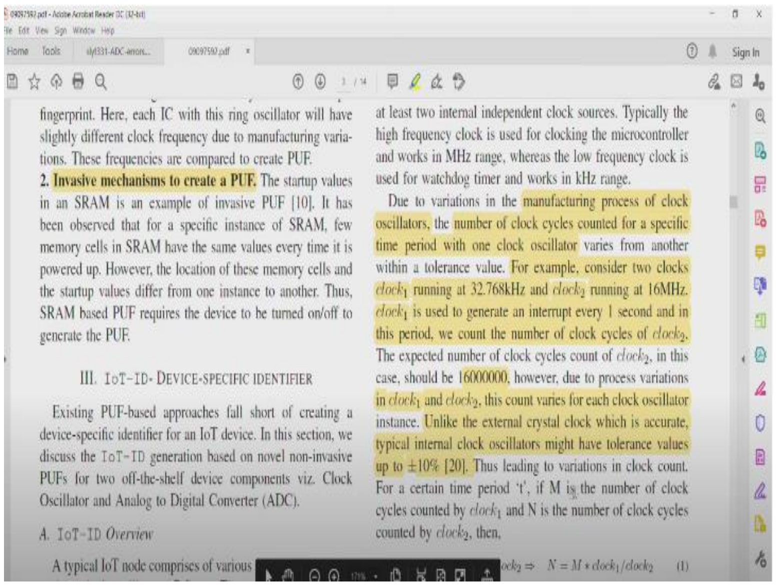
B. Clock oscillator PUF

Every microcontroller needs a clock source as it determines the speed with which the microcontroller operates. Internal or external clock oscillators are used as the clock source for the microcontroller. Most of the present-day controllers such as Atmega2560 [20], CC3200 from Texas instruments [21], nRF52832 from Nordic [22] etc., used in IoT devices have at least two internal independent clock sources. Typically the high frequency clock is used for clocking the microcontroller and works in MHz range, whereas the low frequency clock is used for watchdog timer and works in kHz range.

Due to variations in the manufacturing process of clock oscillators, the number of clock cycles counted for a specific time period with one clock oscillator varies from another within a tolerance value. For example, consider two clocks $clock_1$ running at 32.768kHz and $clock_2$ running at 16MHz. $clock_1$ is used to generate an interrupt every 1 second and in this period, we count the number of clock cycles of $clock_2$. The expected number of clock cycles count of $clock_2$, in this case, should be 16000000, however, due to process variations in $clock_1$ and $clock_2$, this count varies for each clock oscillator

But I already mentioned to you that existing oscillators, existing systems have this problem that they need additional circuitry and therefore they also become invasive by nature. And this paper says, look, you do not have to do any of that we can do without reading anything and it has certain properties and so on.

(Refer Slide Time: 19:46)


fingerprint. Here, each IC with this ring oscillator will have slightly different clock frequency due to manufacturing variations. These frequencies are compared to create PUF.

2. Invasive mechanisms to create a PUF. The startup values in an SRAM is an example of invasive PUF [10]. It has been observed that for a specific instance of SRAM, few memory cells in SRAM have the same values every time it is powered up. However, the location of these memory cells and the startup values differ from one instance to another. Thus, SRAM based PUF requires the device to be turned on/off to generate the PUF.

III. IoT-ID- DEVICE-SPECIFIC IDENTIFIER

Existing PUF-based approaches fall short of creating a device-specific identifier for an IoT device. In this section, we discuss the IoT-ID generation based on novel non-invasive PUFs for two off-the-shelf device components viz. Clock Oscillator and Analog to Digital Converter (ADC).

A. IoT-ID Overview

A typical IoT node comprises of various

at least two internal independent clock sources. Typically the high frequency clock is used for clocking the microcontroller and works in MHz range, whereas the low frequency clock is used for watchdog timer and works in kHz range.

Due to variations in the manufacturing process of clock oscillators, the number of clock cycles counted for a specific time period with one clock oscillator varies from another within a tolerance value. For example, consider two clocks $clock_1$ running at 32.768kHz and $clock_2$ running at 16MHz. $clock_1$ is used to generate an interrupt every 1 second and in this period, we count the number of clock cycles of $clock_2$. The expected number of clock cycles count of $clock_2$, in this case, should be 16000000, however, due to process variations in $clock_1$ and $clock_2$, this count varies for each clock oscillator instance. Unlike the external crystal clock which is accurate, typical internal clock oscillators might have tolerance values up to ±10% [20]. Thus leading to variations in clock count. For a certain time period 't', if M is the number of clock cycles counted by $clock_1$ and N is the number of clock cycles counted by $clock_2$, then,

$$ock_2 \Rightarrow \quad N = M * clock_1/clock_2 \qquad (1)$$

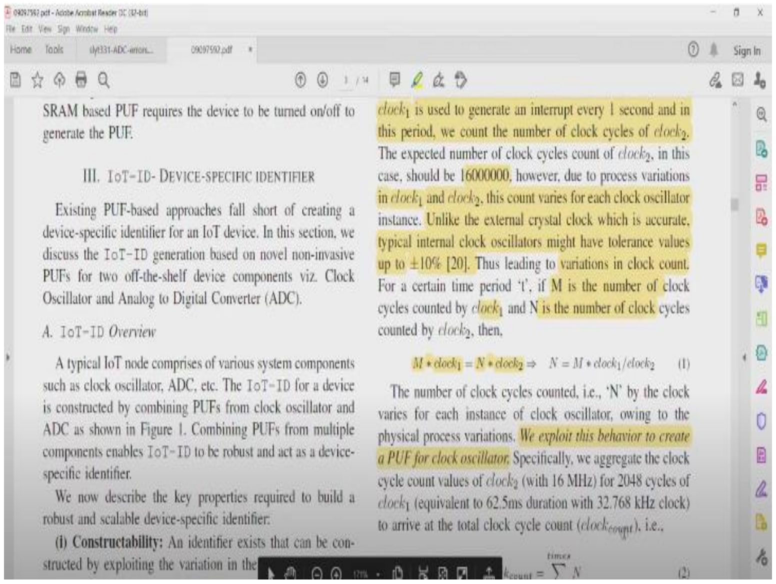
Now what is the crux? Let us get to that. Very simple. Just look at this. Now due to manufacturing variations of clock oscillator, microcontroller, an SoC and IoT device has an ADC as a clock oscillator circuit. Now we are looking at just put your eyeball on the oscillator circuit of the IoT device you purchase from the market. Go and you have bought an Arduino device look at the oscillator of the Arduino board.

That is what you have to put your eyeball on. So let us read it. The number of clock cycles counted for a specific time period with one clock oscillator varies from another with tolerance. I will give you an example. If you consider two clocks, clock 1 running at 32.768 kHz, this is usually the one that is used for timers and clock 2 running at 16 MHz, this is the CPU clock. These are timer clock, and this is the CPU clock. One is called clock one the other is the clock two.

Clock one is used to generate interrupt every one second. And in this period, we count the number of clock cycles of clock two. That is the beauty. You use it like a gate. You use the clock one like a gate. Now you see clock one is used to generate an interrupt every one second and, in this period, we count the number of clock cycles of clock two that is running at a much higher speed. So, what does it mean?

In one second if you are talking about 16 MHz it should give you this number. But it is not going to give you that number. It is going to give you some other number and because of that some other number in clock one and clock two this count varies from each clock oscillator instance. Unlike external clock, which is accurate, typical internal clock oscillators might have tolerance values up to plus minus 10% thus leading to variations in the clock count. So, we are exploiting generation of a PUF using the variations in the clock count.

(Refer Slide Time: 22:10)


SRAM based PUF requires the device to be turned on/off to generate the PUF.

III. IoT-ID- DEVICE-SPECIFIC IDENTIFIER

Existing PUF-based approaches fall short of creating a device-specific identifier for an IoT device. In this section, we discuss the IoT-ID generation based on novel non-invasive PUFs for two off-the-shelf device components viz. Clock Oscillator and Analog to Digital Converter (ADC).

A. IoT-ID *Overview*

A typical IoT node comprises of various system components such as clock oscillator, ADC, etc. The IoT-ID for a device is constructed by combining PUFs from clock oscillator and ADC as shown in Figure 1. Combining PUFs from multiple components enables IoT-ID to be robust and act as a device-specific identifier.

We now describe the key properties required to build a robust and scalable device-specific identifier:

(i) Constructability: An identifier exists that can be constructed by exploiting the variation in the

$clock_1$ is used to generate an interrupt every 1 second and in this period, we count the number of clock cycles of $clock_2$. The expected number of clock cycles count of $clock_2$, in this case, should be 16000000, however, due to process variations in $clock_1$ and $clock_2$, this count varies for each clock oscillator instance. Unlike the external crystal clock which is accurate, typical internal clock oscillators might have tolerance values up to ±10% [20]. Thus leading to variations in clock count. For a certain time period 't', if M is the number of clock cycles counted by $clock_1$ and N is the number of clock cycles counted by $clock_2$, then,

$$M * clock_1 = N * clock_2 \Rightarrow \quad N = M * clock_1/clock_2 \qquad (1)$$

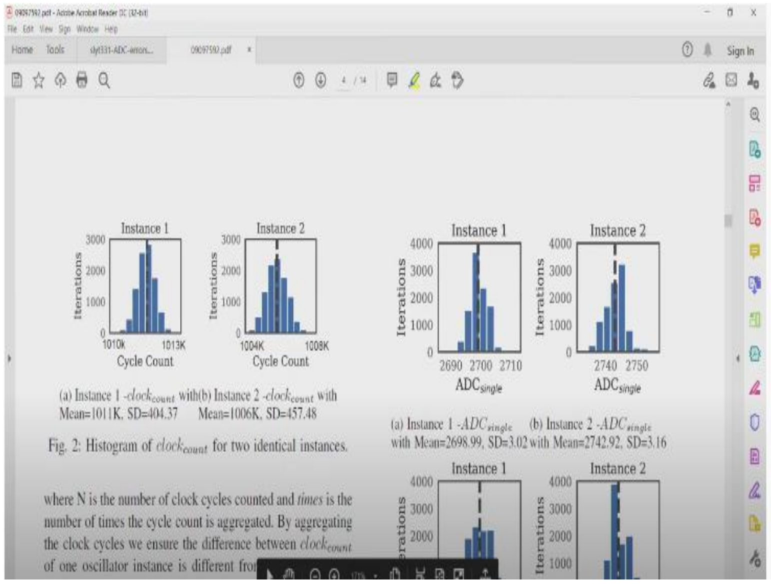
The number of clock cycles counted, i.e., 'N' by the clock varies for each instance of clock oscillator, owing to the physical process variations. *We exploit this behavior to create a PUF for clock oscillator.* Specifically, we aggregate the clock cycle count values of $clock_2$ (with 16 MHz) for 2048 cycles of $clock_1$ (equivalent to 62.5ms duration with 32.768 kHz clock) to arrive at the total clock cycle count ($clock_{count}$), i.e.,

$$k_{count} = \sum^{times} N \qquad (2)$$

And this can be expressed in very simple terms. M is the number of clock cycles counted by clock 1, N is the number of clock cycles counted by clock 2. M times clock 1 is nothing but is to be equated to N times clock 2. So, what is N is the question? It is quite simple. You can see it from here. The N by the clock varies for each instance of clock oscillator. So that is the key. You are saying that look man N is not the same.

It is varying all the time and owing to this physical process variations and keeps changing and therefore we exploit this behavior of PUF oscillators.

(Refer Slide Time: 22:54)



(a) Instance 1 -$clock_{count}$ with Mean=1011K, SD=404.37 (b) Instance 2 -$clock_{count}$ with Mean=1006K, SD=457.48

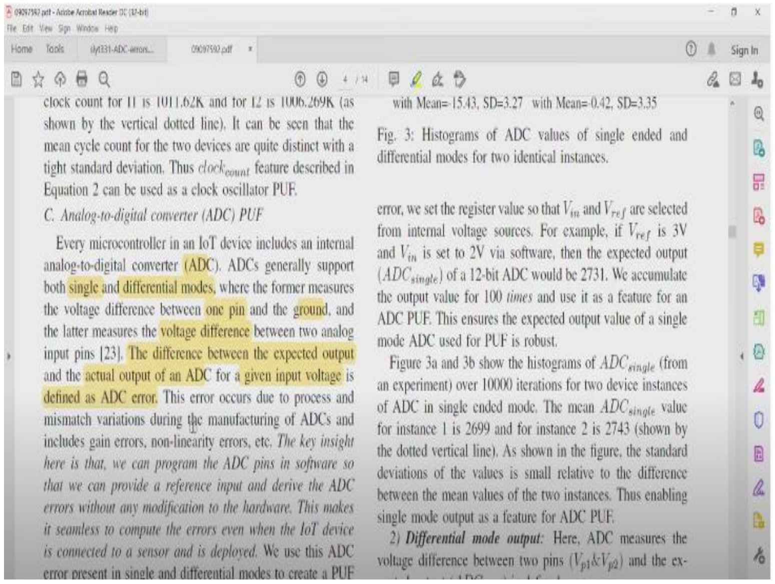
Fig. 2: Histogram of $clock_{count}$ for two identical instances.

(a) Instance 1 -ADC_{single} with Mean=2698.99, SD=3.02 (b) Instance 2 -ADC_{single} with Mean=2742.92, SD=3.16

where N is the number of clock cycles counted and *times* is the number of times the cycle count is aggregated. By aggregating the clock cycles we ensure the difference between $clock_{count}$ of one oscillator instance is different fro

Then there are some tricks that this paper talks about. And then it talks about the instance one and instance two. When you say instance one you are talking about IC 1. Instance two is IC 2. Arduino Mega board 1, Arduino Mega board 2. Look what has happened. Instance one, Arduino Mega board 1, clock count mean is this value, standard deviation is this value. Mean is this value in the second Arduino 2 and the standard deviation is 457 in the other case. So, this is a histogram of clock count for two Arduino boards which is nothing but two instances.

(Refer Slide Time: 23:33)



clock count for I1 is 1011.62K and for I2 is 1006.269K (as shown by the vertical dotted line). It can be seen that the mean cycle count for the two devices are quite distinct with a tight standard deviation. Thus $clock_{count}$ feature described in Equation 2 can be used as a clock oscillator PUF.

C. Analog-to-digital converter (ADC) PUF

Every microcontroller in an IoT device includes an internal analog-to-digital converter (ADC). ADCs generally support both single and differential modes, where the former measures the voltage difference between one pin and the ground, and the latter measures the voltage difference between two analog input pins [23]. The difference between the expected output and the actual output of an ADC for a given input voltage is defined as ADC error. This error occurs due to process and mismatch variations during the manufacturing of ADCs and includes gain errors, non-linearity errors, etc. *The key insight here is that, we can program the ADC pins in software so that we can provide a reference input and derive the ADC errors without any modification to the hardware. This makes it seamless to compute the errors even when the IoT device is connected to a sensor and is deployed.* We use this ADC error present in single and differential modes to create a PUF

with Mean=-15.43, SD=3.27 with Mean=-0.42, SD=3.35

Fig. 3: Histograms of ADC values of single ended and differential modes for two identical instances.

error, we set the register value so that V_{in} and V_{ref} are selected from internal voltage sources. For example, if V_{ref} is 3V and V_{in} is set to 2V via software, then the expected output (ADC_{single}) of a 12-bit ADC would be 2731. We accumulate the output value for 100 *times* and use it as a feature for an ADC PUF. This ensures the expected output value of a single mode ADC used for PUF is robust.

Figure 3a and 3b show the histograms of ADC_{single} (from an experiment) over 10000 iterations for two device instances of ADC in single ended mode. The mean ADC_{single} value for instance 1 is 2699 and for instance 2 is 2743 (shown by the dotted vertical line). As shown in the figure, the standard deviations of the values is small relative to the difference between the mean values of the two instances. Thus enabling single mode output as a feature for ADC PUF.

2) ***Differential mode output:*** Here, ADC measures the voltage difference between two pins (V_{p1}&V_{p2}) and the ex-

Great, you can do almost the same kind of trick you can apply and look at what you can do with ADC. Now we studied ADC in detail. Every controller has this ADC, and you know that an ADC can either connect single ended or you can do differential mode. Both are possible. So, when you say single ended, you are talking about voltage between voltage output and ground and in the differential mode, you are talking about measuring the voltage difference between two analog input pins.

Essentially that is their thing. The difference between the expected output and the actual output of ADC for a given input voltage is defined as the ADC error. And these ADC error occurs due to process and mismatch variations during the manufacturing of the ADCs and includes gain errors, non-linearity errors etcetera. All these errors are there. These issues essentially, the key insight here is that we can program the ADC pins in software so that we can get, we can provide a reference input and derive the ADC errors without any modification to the hardware.

This is amazing that people have problems of input error, ADC error is an issue. And people try to see how to compensate for that error particularly you have to do all those error compensations in software. What is gain error? Gain error is coming because the maximum counter if you take three-bit ADC you go from 000 to 111. And let us say 5 volt is your input, Vref is 5 volts. In other words, your LSB is FSR divided by 2 power n.

FSR divided by 2 power n. That means it is 5 volts divided by 2 power n, n here is 3. Two cubed. So, you will get 111 when the input is 5 volts. Actually, that is the problem. You will not get when your input is 5 volts you would not get 111. You may get it at 4.5 volts, you may get it at 4.3 volts. That is the gain error. That top part is the gain error on the other side. This side also you have a problem. 0 volts is 000. Not true.

Depending on your step size 2 power n you will do. You will know what is your next step up. You take simple example and try. I have given you. You have 5 volts as your reference and n is 3. You know your step size. So, it should come in discrete steps only. So, it will go like this. It will go like a step. But the trouble is it would not go from 0. Your code word 000, 3-bit output should correspond to 0 volts and the next one should come when 001 that means change in one LSB should occur.

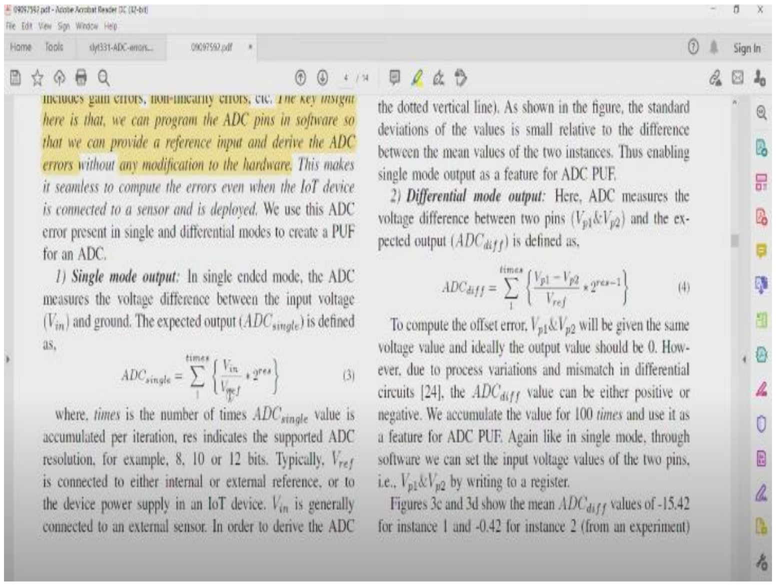
That is what you say as 1 LSB is equal to FSR divided by 2 cubed or 2 power n. It would not come there. It may come early. Let us say it was at 125 millivolt the next step as an example. It may come at 80 millivolt, or it may come after 125 millivolt that step change. It may come at 140 millivolts, it should have come at 125, it may come at 80, 90 or even 100 or it can come after 125. It may come at 130. Again, there is an error, input error.

That is the most fundamental thing about ADCs. So, these two errors are there anyway. You cannot avoid that. You may now ask other questions which we will discuss subsequently. The dynamic range reduces. So many issues will come because you are getting chopped off right in the top. Because you are not able to go up to 5 volts as I discussed earlier. Your gain error has reduced as so high that even before you touched 5 volts the maximum number 111 if it is N is equal to 3 has already come.

Those keep changing for different ICs. From IC to IC, it changes. Why cannot you use that as a signature? These are two I mean; these are correctable errors. By the way this is something that you can correct. In software you can correct. All the dynamic range will reduce you can correct it. But then there are errors which are non-linear. And you cannot even correct them. There is something called integral error, differential error and all that.

These are slightly more advanced things. But you can express them also and use them for generation of ID. So that is the key of this whole discussion about your lookout for what are those little things that I can pull out which I can put together and generate identification for the ID. So that is the point that this paper is actually doing that. Look here, the key insight here is that we can program the ADC pins in software so that we can provide a reference input and derive the ADC errors without any modification to the hardware.

(Refer Slide Time: 29:58)



includes gain errors, non-linearity errors, etc. *The key insight here is that, we can program the ADC pins in software so that we can provide a reference input and derive the ADC errors without any modification to the hardware. This makes it seamless to compute the errors even when the IoT device is connected to a sensor and is deployed.* We use this ADC error present in single and differential modes to create a PUF for an ADC.

1) ***Single mode output:*** In single ended mode, the ADC measures the voltage difference between the input voltage (V_{in}) and ground. The expected output (ADC_{single}) is defined as,

$$ADC_{single} = \sum_{1}^{times} \left\{ \frac{V_{in}}{V_{ref}} * 2^{res} \right\} \quad (3)$$

where, *times* is the number of times ADC_{single} value is accumulated per iteration, res indicates the supported ADC resolution, for example, 8, 10 or 12 bits. Typically, V_{ref} is connected to either internal or external reference, or to the device power supply in an IoT device. V_{in} is generally connected to an external sensor. In order to derive the ADC

the dotted vertical line). As shown in the figure, the standard deviations of the values is small relative to the difference between the mean values of the two instances. Thus enabling single mode output as a feature for ADC PUF.

2) ***Differential mode output:*** Here, ADC measures the voltage difference between two pins (V_{p1}&V_{p2}) and the expected output (ADC_{diff}) is defined as,

$$ADC_{diff} = \sum_{1}^{times} \left\{ \frac{V_{p1} - V_{p2}}{V_{ref}} * 2^{res-1} \right\} \quad (4)$$

To compute the offset error, V_{p1}&V_{p2} will be given the same voltage value and ideally the output value should be 0. However, due to process variations and mismatch in differential circuits [24], the ADC_{diff} value can be either positive or negative. We accumulate the value for 100 *times* and use it as a feature for ADC PUF. Again like in single mode, through software we can set the input voltage values of the two pins, i.e., V_{p1}&V_{p2} by writing to a register.

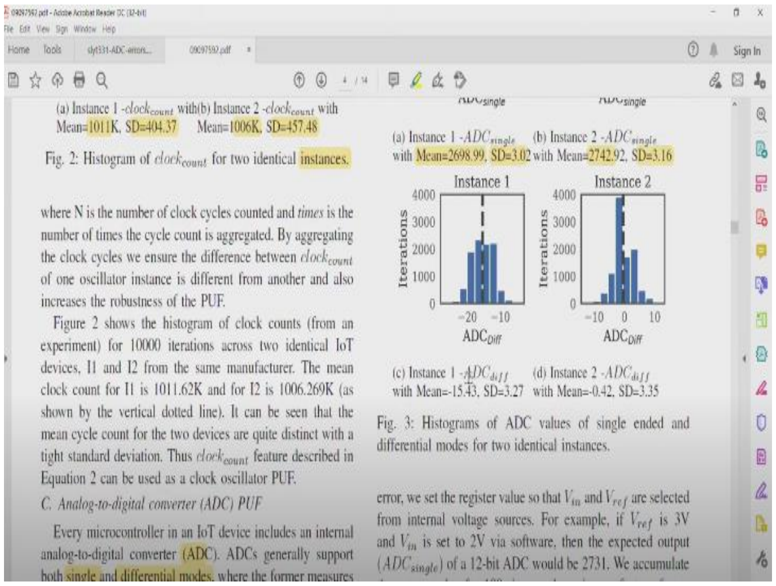
Figures 3c and 3d show the mean ADC_{diff} values of -15.42 for instance 1 and -0.42 for instance 2 (from an experiment)

Now a single ended mode Vin by Vref this is the basic expression for code word generation. Is no t it? Vin by Vref into 2 power n which is nothing but the res here. You do this multiple times and then you do a sigma of that you get ADC single value. But that single value when you start accumulating it many times is different for each chip for the same voltage. Vref is connected to either internal or external reference or to a device power supply in an IoT device.

Vin is generally connected to an external sensor that is right. And look at what has happened. So, this is for the first chip. So let us just look at the results. For the first chip you get the instance here. Instance one ADC with mean of so much and standard deviation of so much and another Arduino chip, another Arduino board mean is 2742 and standard deviation is 3.16. So very different. You can do the same for ADC differential also. And that is easy.

You can see that differential means it will be numerator will not be just Vin, but it will be a difference of input. Vp1 will be one input to the ADC. Vp2 will be the other input. You subtract the two that would be the difference and the Vref is the same. And then of course you do the difference, and you start accumulating it a number of times and you get this kind of a picture.

(Refer Slide Time: 32:00)



(a) Instance 1 -$clock_{count}$ with Mean=1011K, SD=404.37 (b) Instance 2 -$clock_{count}$ with Mean=1006K, SD=457.48

Fig. 2: Histogram of $clock_{count}$ for two identical instances.

where N is the number of clock cycles counted and *times* is the number of times the cycle count is aggregated. By aggregating the clock cycles we ensure the difference between $clock_{count}$ of one oscillator instance is different from another and also increases the robustness of the PUF.

Figure 2 shows the histogram of clock counts (from an experiment) for 10000 iterations across two identical IoT devices, I1 and I2 from the same manufacturer. The mean clock count for I1 is 1011.62K and for I2 is 1006.269K (as shown by the vertical dotted line). It can be seen that the mean cycle count for the two devices are quite distinct with a tight standard deviation. Thus $clock_{count}$ feature described in Equation 2 can be used as a clock oscillator PUF.

C. Analog-to-digital converter (ADC) PUF

Every microcontroller in an IoT device includes an internal analog-to-digital converter (ADC). ADCs generally support both single and differential modes, where the former measures

(a) Instance 1 -ADC_{single} with Mean=2698.99, SD=3.02 (b) Instance 2 -ADC_{single} with Mean=2742.92, SD=3.16

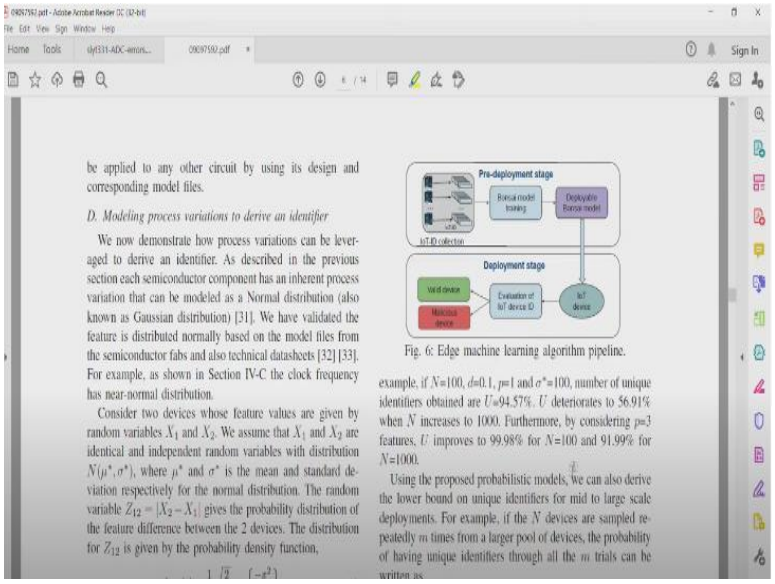
(c) Instance 1 -ADC_{diff} with Mean=-15.43, SD=3.27 (d) Instance 2 -ADC_{diff} with Mean=-0.42, SD=3.35

Fig. 3: Histograms of ADC values of single ended and differential modes for two identical instances.

error, we set the register value so that V_{in} and V_{ref} are selected from internal voltage sources. For example, if V_{ref} is 3V and V_{in} is set to 2V via software, then the expected output (ADC_{single}) of a 12-bit ADC would be 2731. We accumulate

You get instance 1 you see ADC diff is with mean of 15.43 and standard deviation is 3.27. And the here mean is 0.42 and standard deviation is 3.35. So, this is a nice thing that you can go on. And then this paper is talking of so many other things. And it in fact goes into the detail of understanding the physics of the process and so on and so forth.

(Refer Slide Time: 32:34)



be applied to any other circuit by using its design and corresponding model files.

D. Modeling process variations to derive an identifier

We now demonstrate how process variations can be leveraged to derive an identifier. As described in the previous section each semiconductor component has an inherent process variation that can be modeled as a Normal distribution (also known as Gaussian distribution) [31]. We have validated the feature is distributed normally based on the model files from the semiconductor fabs and also technical datasheets [32] [33]. For example, as shown in Section IV-C the clock frequency has near-normal distribution.

Consider two devices whose feature values are given by random variables X_1 and X_2. We assume that X_1 and X_2 are identical and independent random variables with distribution $N(\mu^*, \sigma^*)$, where μ^* and σ^* is the mean and standard deviation respectively for the normal distribution. The random variable $Z_{12} = |X_2 - X_1|$ gives the probability distribution of the feature difference between the 2 devices. The distribution for Z_{12} is given by the probability density function,

Fig. 6: Edge machine learning algorithm pipeline.

example, if N=100, d=0.1, p=1 and σ^*=100, number of unique identifiers obtained are U=94.57%. U deteriorates to 56.91% when N increases to 1000. Furthermore, by considering p=3 features, U improves to 99.98% for N=100 and 91.99% for N=1000.

Using the proposed probabilistic models, we can also derive the lower bound on unique identifiers for mid to large scale deployments. For example, if the N devices are sampled repeatedly m times from a larger pool of devices, the probability of having unique identifiers through all the m trials can be written as

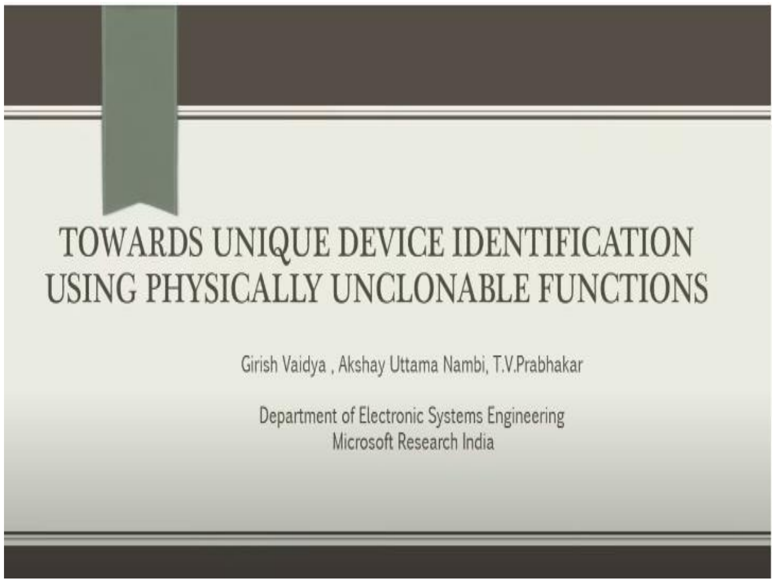
Then you can also take that and then because we have to show a complete system implementation, you take signatures from 50 odd devices, or 70 odd devices or 100 odd devices, put it into a model, train the model, and then use a classifier and then actually identify the ID. So, this paper is going into that kind of detailing. So folks, the point which is emerging now is that clock oscillators can essentially be exploited on the IoT device and so can ADC components be exploited from the ADC device.

You can try this in your homes. Let us now shift to a very more lot more exciting thing which is the demo part. What I will show you in the demo in fact Abishek, our project staff is here, what we will show you is neither ADC nor oscillator you can also generate PUFs from SRAMs from the RAM. You may ask how is that possible? It is another strange story here that it turns out 20% of the RAM locations on startup, very important. Please listen carefully.

On startup always skew to either zero or one each time. They are skewed, finished. If it is skewed to 1, it will be 1 all the time. If it is skewed to 0, it will be 0 all the time. So now you have a fantastic situation that if it is skewed like this and even if I switch off and switch on, I will get back the same ID if I know which are the cells in RAM which are skewed. All I had to do is read those.

Question is how many of them? As I mentioned about 20% of them that is on the higher side. Even 10% is good enough for you folks to generate a good device ID. That is the demo we should see, appreciate and get a feel for the problem. So let us run through a slide deck in a concise way so that you will get an understanding of physically unclonable functions.

(Refer Slide Time: 35:19)



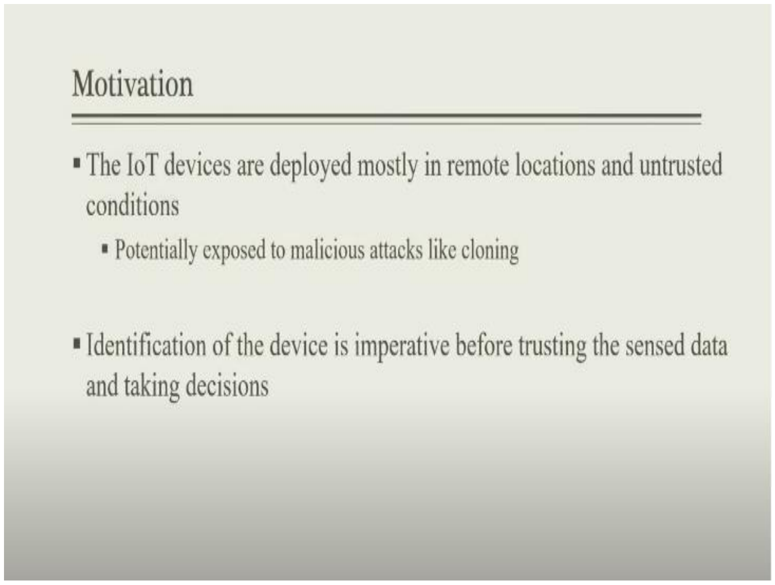
TOWARDS UNIQUE DEVICE IDENTIFICATION USING PHYSICALLY UNCLONABLE FUNCTIONS

Girish Vaidya , Akshay Uttama Nambi, T.V.Prabhakar

Department of Electronic Systems Engineering
Microsoft Research India

So, this work is the IoT ID paper that we read up together.

(Refer Slide Time: 35:22)

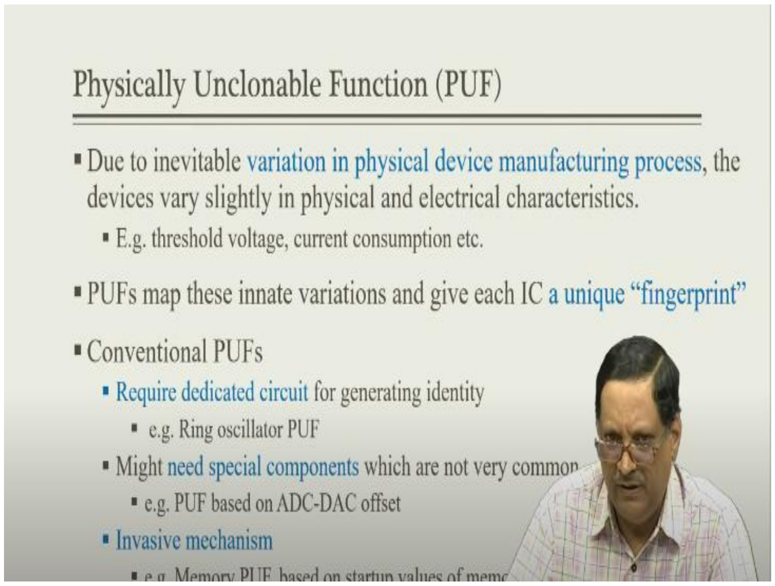


Motivation

- The IoT devices are deployed mostly in remote locations and untrusted conditions
  - Potentially exposed to malicious attacks like cloning
- Identification of the device is imperative before trusting the sensed data and taking decisions

And you can see that this work was done in collaboration with Microsoft Research India. So, motivation is here. They are mostly deployed, their IoT nodes are mostly deployed in remote locations and identification of these devices is imperative before trusting the sensed data and taking decisions.

(Refer Slide Time: 35:42)



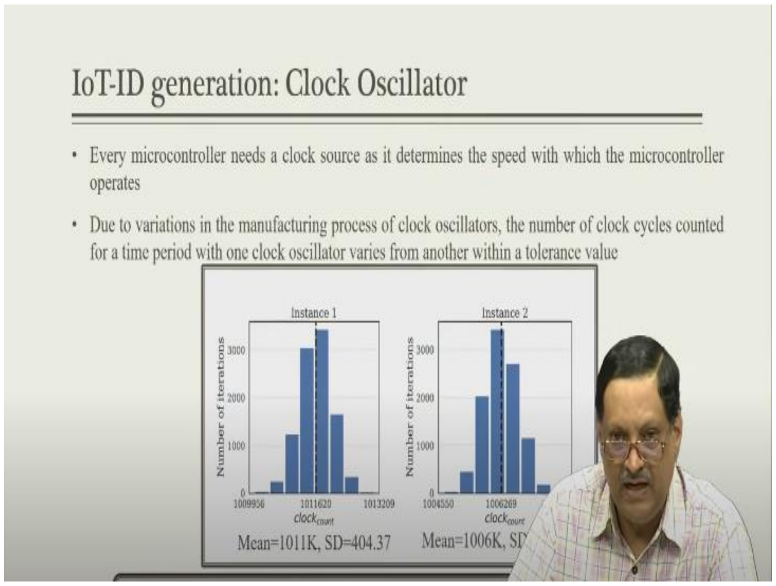
Physically Unclonable Function (PUF)

- Due to inevitable variation in physical device manufacturing process, the devices vary slightly in physical and electrical characteristics.
  - E.g. threshold voltage, current consumption etc.
- PUFs map these innate variations and give each IC a unique "fingerprint"
- Conventional PUFs
  - Require dedicated circuit for generating identity
    - e.g. Ring oscillator PUF
  - Might need special components which are not very common
    - e.g. PUF based on ADC-DAC offset
  - Invasive mechanism
    - e.g. Memory PUF based on startup values of memo

And now we are looking at inevitable variation in physical device manufacturing which could essentially be exploited. It includes threshold voltage, current consumption and all that. And PUFs may map these small variations into something which has a very unique fingerprint. Conventional PUFs are there because PUF is an old area of research and people have used it. Ring oscillator PUF is a very old one.

Then there might be special components which are not very common. You can have several IoT devices that do not have DACs essentially. So, which essentially means that you cannot use all types of components inside the device for signatures and then you also have problems, you cannot invade into it, you cannot get a device from the market and then start getting a PUF out of it. That is also not something that you can do. I mentioned this already.

(Refer Slide Time: 36:52)

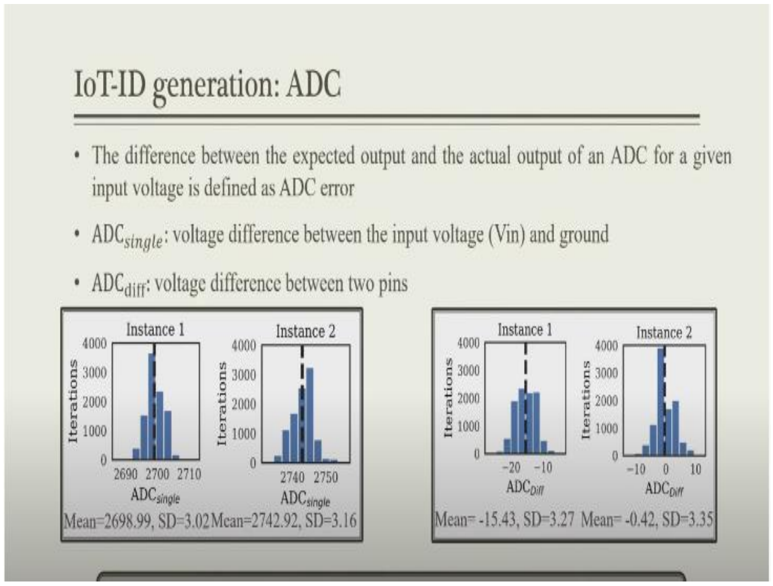


IoT-ID generation: Clock Oscillator

- Every microcontroller needs a clock source as it determines the speed with which the microcontroller operates
- Due to variations in the manufacturing process of clock oscillators, the number of clock cycles counted for a time period with one clock oscillator varies from another within a tolerance value

These are some very simple results. I discussed this; I am just showing you the slides so that you get a feel for it. This is a mean clock count for two devices are quite distinct. You can try this over 50 devices, hundreds and hundreds of devices. You will find that they are each one quite different.

(Refer Slide Time: 37:10)

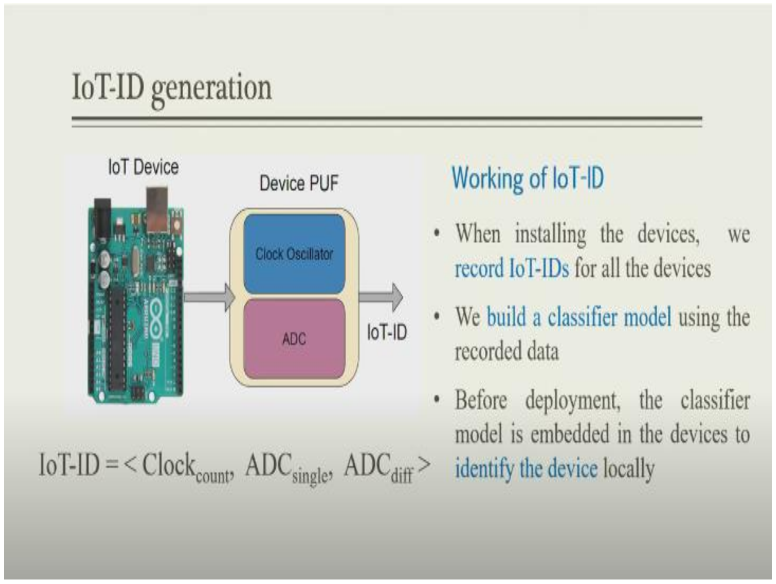


IoT-ID generation: ADC

- The difference between the expected output and the actual output of an ADC for a given input voltage is defined as ADC error
- ADC_{single}: voltage difference between the input voltage (Vin) and ground
- ADC_{diff}: voltage difference between two pins

That is from clock oscillator, and this is from ADC. And you can see that the ADC device one and device two are different, the mean and the standard deviation is different for the ADC output. And so, you can do this for single ended as well as for differential modes. ADC single mode and differential mode for the two devices are quite distinct again.

(Refer Slide Time: 37:36)


IoT-ID generation

$IoT\text{-}ID = < Clock_{count}, ADC_{single}, ADC_{diff} >$

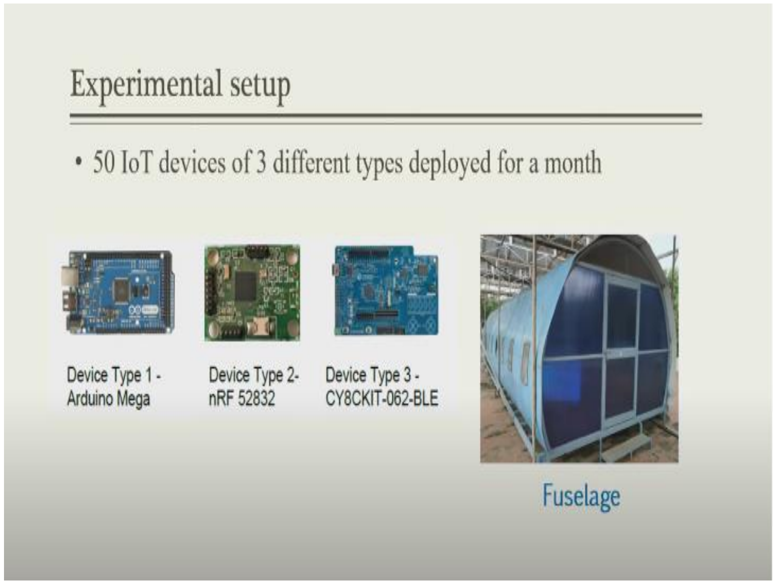
Working of IoT-ID

- When installing the devices, we record IoT-IDs for all the devices
- We build a classifier model using the recorded data
- Before deployment, the classifier model is embedded in the devices to identify the device locally

And you can use IoT ID can be generated from clock count as well as ADC single and ADC differential. Very similar to Aadhar ID you do not take just the biometric. You take a fingerprint, and you use biometric, and you use name, you use physical identification, so many things and then generate the ID. So, it is not one component, but it is many of them. So quite like then they take the device ID that we build the classifier.

And the before deployment the classifier model is embedded into the devices to identify the device locally.

(Refer Slide Time: 38:20)

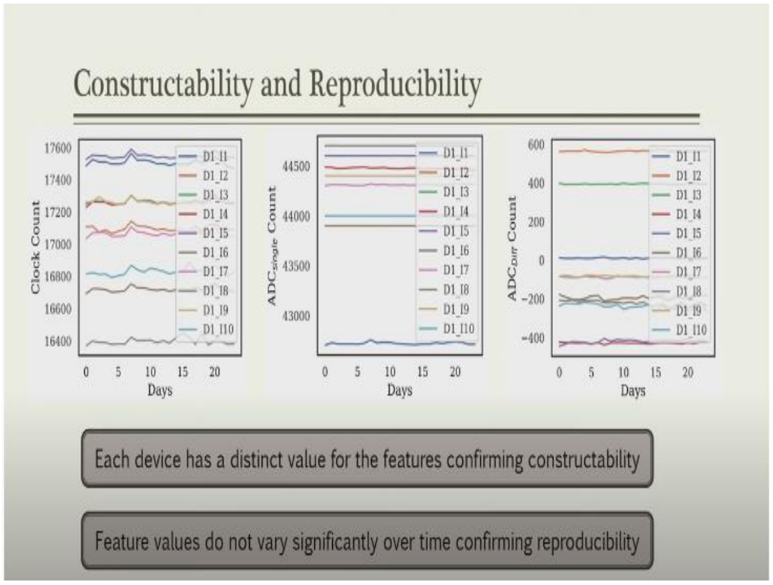

Experimental setup

- 50 IoT devices of 3 different types deployed for a month

Device Type 1 - Arduino Mega
Device Type 2- nRF 52832
Device Type 3 - CY8CKIT-062-BLE
Fuselage

Essentially, it is something that you can actually try. It was tried out on 50 devices, Arduino Mega, nRF52832 as well as Cypress circuit and they put it onto an environment in which this could be tried.

(Refer Slide Time: 38:33)



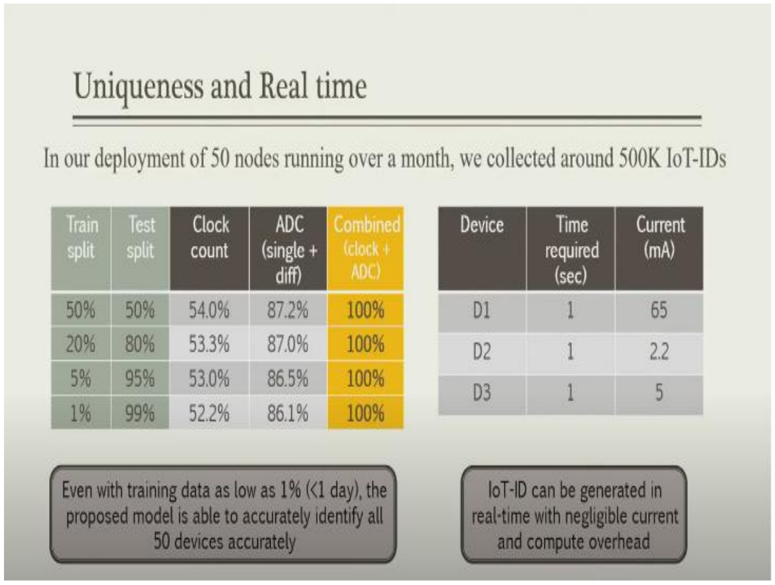
Constructability and Reproducibility

Each device has a distinct value for the features confirming constructability

Feature values do not vary significantly over time confirming reproducibility

And these are some nice results. You can see that the clock count how it is varying over days. Idea is it should be as straight as possible so that you maintain uniqueness. ADC count also are unique lines, distinct lines over a number of days. Each device has a distinct value for features conforming to constructability. Feature values do not significantly vary over time also produces the reproducibility.

(Refer Slide Time: 39:02)



Uniqueness and Real time

In our deployment of 50 nodes running over a month, we collected around 500K IoT-IDs

Train split	Test split	Clock count	ADC (single + diff)	Combined (clock + ADC)
50%	50%	54.0%	87.2%	100%
20%	80%	53.3%	87.0%	100%
5%	95%	53.0%	86.5%	100%
1%	99%	52.2%	86.1%	100%

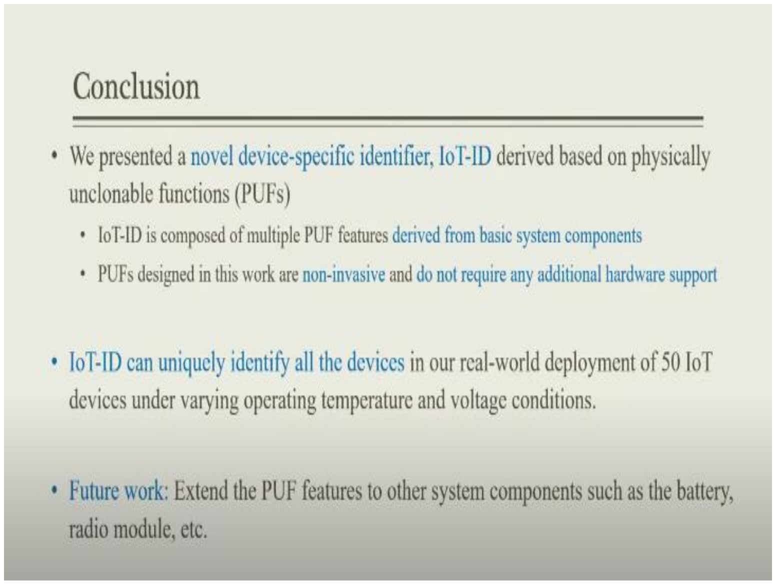
Device	Time required (sec)	Current (mA)
D1	1	65
D2	1	2.2
D3	1	5

Even with training data as low as 1% (<1 day), the proposed model is able to accurately identify all 50 devices accurately

IoT-ID can be generated in real-time with negligible current and compute overhead

So, these are some additional numbers to give you an idea. 50 nodes were running over one month, and 500,000 IoT IDs were collected, they were processed. And then you can see that when you do a combination of clock plus ADC you get 100% uniqueness. So that is another thing. And this is some idea related to the fact that it is not very expensive energy wise. You can generate in real time with negligible current and compute overhead.

(Refer Slide Time: 39:37)



Conclusion

- We presented a novel device-specific identifier, IoT-ID derived based on physically unclonable functions (PUFs)
  - IoT-ID is composed of multiple PUF features derived from basic system components
  - PUFs designed in this work are non-invasive and do not require any additional hardware support
- IoT-ID can uniquely identify all the devices in our real-world deployment of 50 IoT devices under varying operating temperature and voltage conditions.
- Future work: Extend the PUF features to other system components such as the battery, radio module, etc.

And that is the idea. So, the idea of that paper was it presented a novel device ID based on physically unclonable functions and it used basic system components like clock and ADC. And you do not need to be invasive by nature. You can extract this quite simply. The whole experiment was tied with 50 IoT devices under varying temperature and voltage conditions. And of course, we have not tried for 500 and 5000 and so on. But I am sure that kind of repeatability will come. It is not that it is a silver bullet.

For all device ID related solutions, folks, do not ever look at it that way. It has limited scope. So, you have to note every solution will have a limited scope. For example, there are no results with respect to variations in temperature. What about aging? Once you put in our case, the fingerprint ID and all that human thing over decades, it is still the same. But what is aging in silicon? We do not know. Nobody has studied that.

Even the tests have not indicated any accelerated aging that was performed. So disclaimer, please note that aging has not been tried and tremendous variations in temperature also have not been studied. But temperature you can handle it. It is not that it is going to be impossible to tame that variation in temperature. Because if variation in temperature occurs it is obvious all devices which are in the vicinity also will rise together.

So, it is not that you cannot compensate for temperature, but aging is difficult. So, one has to research, and one has to understand that these are all things that will have to evolve over the years and people will find solutions. On a closing note, the area of quantum is picking up. So, a lot of work related to quantum happening. You can also be talking about quantum PUFs. These are small leakage currents which are caused in the oxide layer during the manufacturing process.

So, we can be talking about quantum PUFs as well. All for future study. This is just to excite you that you can generate unique IDs without too much of any innovation from simple systems using commercially available components. And it is a very important area when you want to talk about device ID for the reasons that we stated. So now let us look at the demo part. This demonstration will actually not be around either the clock oscillator PUF or this one, the ADC single ended and differential.

But we look at the SRAM PUF. So that is the big takeaway. How to get to the SRAM PUF is the real demonstration which you can also try back in your homes, in your labs and so on and so forth. What I have here is this board.
(Video Starts: 42:46)

This board is a development board. This is a development board from Nordic. You can see here; it is written here Nordic. This is a Nordic processor called 52832. This is the series 52832 processor series. It contains a 64MHz cortex M4 with FPU that is floating point unit. It has RAM of 32 kilobytes. It has a flash of 256 kilobytes. It supports multi radio, it supports in 2.4 gigahertz it supports two radios. One it supports Bluetooth 5.0, and it also supports 802.15.4 ZigBee Radio.

So, both protocols it supports. Anyone you can run. But essentially it can support both of them. And therefore, this is amazing thing. So, they are good when you are a front end which can work with either with the Bluetooth part of the protocol stack or IEEE 802.15.4 ZigBee stack. So, it is really amazing. Lot of integration there are ADC ports, then there is I2C, then there is SPI. So, it is very powerful. As a chip it is a very powerful integrated System on Chip, SoC it is called.

Now our objective is can you pull out the SRAM PUF from this particular chip. For that let us start the demonstration and let us connect this particular board to a laptop and I have this laptop here. So now let us move to this laptop, we will connect on serial port too this board. And you can see that we have invoked, the serial port terminal session and now we will open that. So, it is possible that you have to identify the rate com port and then connect to the system.

Now you see the screen is zooming all along the code that is fused on the system is for generating the PUFs. What we see on the extreme rate is all ones being written to about 1 kilobyte of locations. Now each location when I say is a double word. In other words, it comprises of 4 bytes. So that is 32 bits in each location. So that is the organisation of our code. So, we are writing 11111, 32 ones into location 1, then writing 32 ones in location 2.

Like that we are writing to 1024 locations with just 1 kilobyte of locations. First part, that means we have switched on and we are writing all ones. Now let us see what we should do next. We switch off the system and we will see what exactly happens on startup of this. Now you see it has started again and on startup these are the vague set of values. On all these startups we have identified 1024 locations. They are all different which is good.

That is what you would get on startup. See that pause which happened in the middle actually shut down the board and it has restarted the board. That means everything written on the RAM is removed. So, what we wrote was all FFFF. Everything got erased because it got shut down and you got this random set of values. Only concentrate on the extreme right, what you see on the extreme right.

So, we are continuously writing the 1024 locations and each location is a double word which is 4 bytes long. So let us see. This process goes on and on and on. This is the round 1 we got this. We need to save it somewhere and you need to run it again. That is, you shut down the machine and again run it and get another round of writing. Then you store all of that and again you start and again you run.

So, it depends on the number of times you want to check. So, you can see now it is shut down and it has done this writing repeatedly and then now we will process this data by taking all these values of multiple rounds into an excel sheet perhaps and then we will start looking up the contents of each one of these locations. Now let us shift our focus to the processing part.
(Video Ends: 48:46)

(Video Starts: 48:47)
Now what you see on the extreme left side are all the locations. You can see that we have taken cycle 1 then you leave the location in cycle 1 then you shut down again you read these 1024 locations in cycle 2, cycle 3, and cycle 4 and so on. You see these cycles. So, you see the very first column cycle 1, cycle 2, cycle 3 and so on. It goes on like this. You can try any number of times. So, we took it for some number of cycles and now we will look at the locations.

What is the summary of what actually happened? What you see are 4 additional columns, BT, BU, BV, and BW. These are the 4 columns, and we are looking up these 4 columns. Why these 4 columns? Essentially these 4 columns correspond to the bytes which are there in each location. How many bytes are there in each location? We said there are 4 bytes. That is why you have 4 columns. If the first byte matches you will get from across all the cycles if it matches you get a true.

If the second byte matches along with the fist byte you will get a true again. If the third byte matches along with the fist byte and second byte you will get true in the third column. If all the 4 bytes match that is byte1, byte 2, byte 3 and byte 4 then you will get true, true, true, true in all locations. Here these numbers you can see that 1024 locations which are to be shown are displayed here. So, you can see that the total count is 1025.

Essentially the first row has to be neglected which is basically the heading. So, 1024 is beautifully shown here. What you also see is the first byte if it is not matching across the 10 cycles it is showing false, the second byte, the third byte and the fourth byte and so on. Somewhere randomly you can all see that the second byte across 10 cycles is actually true. This is true, this is false and so on. So, it is all in random order.

So, you may actually want to see how many first byte locations are actually repeating. Let us do a simple sort and then check whether the first byte location how many of them are true. So that filter if you apply you will see everything true here and you see that count as 325. So already out of 1024, if you consider 1024 locations each location being 32 bits, 8 bits match across the cycles, 1 byte match across all these locations are 325.

Now let us see what happens if the first and the second that means you are putting a tight up constraint. You are looking at first byte and the second byte, both of them should match. In all of these 1024 locations how many of them actually match? Just look at the number and then we will see that. Obviously, it would not be 1024. It would not even be 325 because 325 is already a big match. It has to be something less than 325.

So let us look at that number. That number turns out to be 106. That means the first byte and the second byte both of them are matching. you get 106 locations where the first and second byte match. Now let us put another constraint. We are interested in the third byte also matching. Obviously, it has to be less than 106 and let us see what is the number. And that number turns out to be 35.

Now let us do for all the locations. You can keep looking at the excel sheet you will see true in all columns and now let us see how that will be how many locations will actually match. You should get something less than 35 and roughly you actually get 17 locations.

(Video Ends: 54:05)

So, summary of this whole exercise we can share this with you, and you can try this excel file, you can use this excel file. I can share the source code of what has been written here to prove to yourself that this is a very prudent way of generating an ID. We did nothing here. We just rebooted and pulled out those locations which are repeating each time because of the skewness of the cells. So that is what we ultimately are interested in.

So, the exercise summary is that one can generate physically unclonable functions for IoT devices in non-invasive and partially invasive way. Because if you want to get to the location you want to see how many are matching you have to reboot and if it is a running program, it is not going to give you anything because it is already written because these are RAM locations. So, in order to generate the ID if someone asks the ID then we have to reboot then pull out the signatures and give to the person that has asked.

Therefore, this is partially invasive although not as much as the ring oscillator PUFs and all that. Special circuits are actually deployed inside. In fact, you can also look up another link called the dielet. Now people are trying to put a lot on dielets. Just Google for dielet you will find dielets as part of the whole system whole way of embedding device ID into components. Thank you very much.