CyberRadar: A PUF-based Detecting and Mapping Framework for Physical Devices

Dawei Li, Di Liu, Yangkun Ren, Ziyi Wang, Yu Sun, Zhenyu Guan, Qianhong Wu, and Jianwei Liu
Beihang University

Abstract—The core issue of cyberspace detecting and mapping is to accurately identify and dynamically track devices. However, with the development of anonymization technology, devices can have multiple IP addresses and MAC addresses, and it is difficult to map multiple virtual attributes to the same physical device by existing detecting and mapping technologies.

In this paper, we propose a detailed PUF-based detecting and mapping framework which can actively detect physical resources in cyberspace, construct resource portraits based on physical fingerprints, and dynamically track devices. We present a new method to implement a rowhammer DRAM PUF on a general PC equipped with DDR4 memory. The PUF performance evaluation shows that the extracted rowhammer PUF response is unique and reliable on PC, which can be treated as the device’s unique physical fingerprint. The results of detecting and mapping experiments show that the framework we proposed can accurately identify the target devices. Even if the device modifies its MAC address, IP address, and operating system, by constructing a physical fingerprint database for device matching, the identification accuracy is close to the ideal value of 100%.

Index Terms—detecting and mapping, DRAM physically unclonable function (PUF), physical fingerprint, rowhammer PUF

I. INTRODUCTION

Cyberspace detecting and mapping refer to detecting various resources in cyberspace, acquiring various attributes of resources, analyzing the relationship between resources, and presenting resource attributes and analysis results in the form of logical topologies or Internet maps. The purpose of detecting and mapping is to obtain the attributes and overall status of cyberspace resources, understand the interactions and development trends between resources, and perceive the dynamic changes in cyberspace.

The object of detecting and mapping is all kinds of physical resources and virtual resources in cyberspace, involving a lot of technologies. The detection range can be the entire Internet, a certain local area network, or a specific network range. Some previous work [1]–[3] investigated and summarized the relevant concepts and technical routes in the detecting and mapping of cyberspace resources, but they did not implement specific detecting and mapping examples.

The traditional resource detection technology obtains information such as the IP address and port number of the target resource by sending a request packet to the target resource. Mature tools that are widely used include Nmap [4], Zmap [5], Masscan [6], Shodan [7], and so on. These detection tools focus on the network layer and cannot obtain the unique physical characteristics of physical devices. Once the device changes its virtual attributes such as its IP address, it will be treated as a new device, and there is no way to continuously track the target device.

Different from the previous work, our detecting and mapping framework focuses on the physical layer. We conduct actual detecting and mapping in a local area network environment, and the target of the detection is the physical resources in the network. The framework we propose can extract physical fingerprints of resources, construct portraits of resources, obtain behavioral relationships between resources, and accurately identify and track target resources.

Our detecting and mapping framework also brings new methods for dark web detection. Some devices are configured with multiple network interface cards and are connected to the Internet and the dark web at the same time. The devices have different virtual attributes in different networks and show different identities. It is difficult for us to identify such devices through traditional detection methods. However, since the device does not replace the memory module every time it connects to a different network, so its physical fingerprint based on the memory module will not change. We can detect on the Internet and the dark web separately, and identify such devices at the border of the Internet and the dark web by comparing unique physical fingerprints.

A. Challenges

Physical Fingerprint Generation. There are two difficulties in remotely detecting and acquiring physical fingerprints of physical resources in cyberspace. The first is the remote acquisition because the detection target is unknown and cannot be physically touched. A reliable method of remote code execution and data return is required to complete the detection. The second is to obtain a physical fingerprint that can uniquely identify the device, and the physical fingerprint is required to be stable and reproducible. The process of generating physical fingerprints should not affect the normal operation of the target device, such as causing the target device to restart or service abnormalities.

Device Dynamic Identification and Tracking. After completing detecting and mapping, the portrait and topological relationship of the resources can be obtained. However, data flows and resource changes are constantly occurring in cyberspace, and resources need to be periodically detected to update data and grasp the development trend of cyberspace. Virtual attributes such as IP address and MAC address of the target device can be replaced at will just like a vest. When the
target device reappears in cyberspace after changing the vest, we hope to be able to identify it through physical fingerprints and continue to track it.

B. Our Contributions

- We propose a detecting and mapping framework based on physical unclonable function (PUF). The physical fingerprint of the target device is extracted through the designed PUF algorithm. The device’s physical fingerprint and other attributes are combined to form its portrait, and then the detection results are visualized in the form of a knowledge graph.
- We implement a rowhammer-based DRAM PUF that can be accessed at runtime on a personal computer running the Linux operating system and equipped with DDR4 memory. It only needs to access the memory module to run the PUF implementation code and does not require additional hardware.
- We evaluate the performance of rowhammer PUF under two metrics. The evaluation results show that the implemented PUF exhibits near-ideal uniqueness and good reliability, which can perfectly distinguish different devices. To the best of our knowledge, our work is the first to implement and evaluate rowhammer PUF on a personal computer.
- We conduct three detecting and mapping experiments in a local area network environment. The experimental results show that even if the virtual attributes such as the MAC address, IP address, or even operating system of the target device are replaced, the target device can be identified and tracked through the physical fingerprint.

II. BACKGROUND

A. Detecting and Mapping

Cyberspace detecting and mapping is a process of detection, analysis, visualization, and application. First, obtain data on cyberspace physical and virtual resources through various detection techniques. Secondly, analyze these data to obtain knowledge of resource attributes and behaviors. Then, visualize the detection data and analysis results. Finally, it can be applied in different scenarios based on the results of detecting and mapping.

- **Detection.** Detection is the process of obtaining data of cyberspace resources, such as the attribute values of resources. The purpose is to provide basic knowledge for the subsequent analysis process. For example, obtain the target device’s IP address, port number, traffic packet, and other data through network scanning software.

- **Analysis.** Analysis is the process of extracting resource attributes from the results returned by the detection process, identifying resources, and constructing resource portraits. The purpose is to form a knowledge base of cyberspace resources.

- **Visualization.** Visualization is the process of constructing network topology and performing visual displays based on the attributes and mapping relationships of cyberspace resources. The purpose is to dynamically display the distribution and status of cyberspace resources.

**Cyberspace resources.** The objects of detecting and mapping are cyberspace resources, including various types of physical resources and virtual resources. Physical resources include switching equipment such as routers, switches, base stations, and access equipment such as mobile phones, computers, and cameras. Virtual resources include virtual services such as DNS and mail, virtual content such as videos and messages, and social media accounts.

**Resource attributes.** Cyberspace resources have specific attributes, which reflect the characteristics of resources and the relationship between resources. Resource attributes can be divided according to different levels, such as the physical layer, logical layer, and cognitive layer. The physical layer reflects the physical attributes of the resource, such as coordinates, dimensions, and specific information about the device. The logical layer reflects the network attributes of resources, such as IP, domain name, port, operating system, etc. The cognitive layer reflects the social attributes of resources, such as service type, service object, and organization.

B. Rowhammer

Rowhammer is a DRAM memory vulnerability, which has been extensively studied by researchers since 2014. When a row (aggressor row) in the memory is repeatedly and rapidly accessed, some bits of its adjacent rows (victim rows) will be flipped. The root cause of this phenomenon is that repeated access to a certain DRAM row will accelerate the charge leakage of adjacent rows.

To achieve rapid access to the rows in the DRAM, the cache mechanism between the CPU and the memory needs to be bypassed. Commonly used methods include flushing the cache using `clflush` or `clflushopt` instructions, cache eviction, uncached memory, and so on.

To successfully induce bit flips in DRAM, the first step is to get the mapping relationship between virtual addresses to physical addresses and then to DRAM addresses (a specific channel, DIMM, rank, bank, row, and column). Then, we need to determine the appropriate hammering patterns to implement reliable bit flips in DRAM.

![Fig. 1. DRAM organization.](image-url)
**DRAM organization.** As shown in Figure 1, the memory controller in the CPU sends a memory request to the DRAM rank through the channel and one channel can carry multiple DRAM modules (DIMMs). A DRAM module has one or more DRAM ranks and a rank generally has 8 DRAM chips. A DRAM rank has multiple banks and one bank spans multiple chips. DRAM banks are further divided into rows and columns of memory cells, where each cell consists of a transistor and capacitor. For true cells, a fully charged capacitor represents a bit of data “1”, and the opposite is true for anti cells. The memory controller is responsible for converting physical addresses into corresponding channels, DIMMS, ranks, and banks in the memory.

**Reverse Engineer the Address Mapping.** The mapping of virtual addresses to physical addresses is implemented through the operating system, such as accessing the pagemap interface or using huge pages. However, not all the mapping relationships between physical addresses and DRAM addresses are public. AMD discloses the address mapping function but Intel does not. For Intel’s CPU, we need to reverse engineer the memory address mapping function.

DRAMA and DRAMDig are two software-only DRAM address mapping reverse engine tools. They all use the time side channel to determine the bank bits. Specifically, each bank in DRAM has a row buffer, which is used to access a row in the bank. Randomly select some address pairs, access them repeatedly and alternately, and measure the average access time. If a pair of addresses are in the same bank, their access time will be significantly higher than those address pairs that are not in the same bank. This phenomenon is also called row buffer conflicts, so we can find an address set whose addresses are all in the same bank and determine the bank bits in the address.

**Hammering patterns.** Hammering patterns refer to the combination of different aggressor rows, which have a great influence on the probability of bit flips. By selecting a suitable hammering pattern, more bit flips can be obtained. There are currently four main hammering patterns:

- **One-location Rowhammer and Single-sided Rowhammer.** One-location hammering pattern, which repeatedly accesses a single row in the bank. This pattern only opens one row repeatedly, so it does not cause row buffer conflicts. Although this pattern induces a few bit flips, it can bypass the protection mechanism based on memory access pattern analysis in some cases. Single-sided hammering pattern just repeatedly accesses two rows in the same bank, and there is no specific positional relationship between the two rows.

- **Double-sided Rowhammer.** Double-sided hammering accesses two rows that satisfy the adjacency relationship. The two rows are adjacent to the same victim row, which can increase the probability of inducing bit flips. For most DDR3 DRAMs that lack in-DRAM Target Row Refresh (TRR) mitigation mechanisms, single-sided or double-sided hammering can induce bit flips. But for the new DDR4 DRAM with TRR mechanism, it is difficult for these two patterns to induce bit flips in practice.

- **Many-sided Rowhammer.** This pattern has the same adjacency relationship between two aggressor rows as the double-sided pattern. Frigo et al. first presented a many-sided rowhammer tool, which can automatically identify hammering patterns and induce reliable bit flips on DDR4 DRAM with in-DRAM TRR mitigation. Since the TRR sampler can only track a limited number of aggressor rows, once the number of aggressor rows exceeds the size of the TRR sampler, the attacker may successfully induce bit flips.

Moreover, many previous works have shown that most of the bit flip locations in DRAM are stable, and random differences in the manufacturing process make the bit flip locations different in different DIMMS of the same design. Therefore, the rowhammer effect can be used to implement an intrinsic PUF.

### III. PUF-BASED DETECTING AND MAPPING FRAMEWORK

One goal of detecting and mapping is to dynamically track physical devices in cyberspace. Specifically, even if the device’s IP address, geographic location, or even the operating system is changed, it can also be identified.

The way we achieve this goal is to obtain the physical fingerprint of the device and bind its physical fingerprint to other attributes of the device. We design an algorithm and implement a rowhammer PUF in the DRAM of the device to obtain the unique physical fingerprint of the device. The detecting and mapping framework is shown in Figure 2.

![PUF-based detecting and mapping framework](image)

**A. PUF for Fingerprint Generation**

The function of PUF in the detecting and mapping framework is to assign a unique physical fingerprint to physical resources in cyberspace. The PUF response is obtained by sending the constructed PUF challenge to the physical device in cyberspace. The physical fingerprint of the device is formed by the obtained PUF response.

Since the detected cyberspace resources are unknown and cannot be physically accessed, those PUFs based on FPGA dedicated circuits cannot meet the actual needs of detecting and mapping, such as the arbiter PUF. A suitable choice is to implement PUF through the components of the cyberspace resources such as SRAM and DRAM, which are widely used in Internet devices, especially personal computers.
The elliptical node in the graph represents a resource entity, topology can be visualized in the form of a knowledge graph. PoP (Point of Presence) level topology discovery. The resource layer can be divided into IP interface level, router level, and all the attributes of the resource into one portrait.

are not dividing the attributes according to levels, but putting topological relationship between physical resources. Here we resources, and virtual attributes are used to describe the Physical fingerprints are used to uniquely identify physical be modified, so they can also be called virtual attributes. resource portrait, only the physical fingerprint will not be

described using the information shown in Figure 3. In the C. Resource Portrait and Topology Analysis

The portrait of physical resources in cyberspace can be described using the information shown in Figure 3. In the resource portrait, only the physical fingerprint will not be changed. Except for physical fingerprints, other attributes can be modified, so they can also be called virtual attributes. Physical fingerprints are used to uniquely identify physical resources, and virtual attributes are used to describe the topological relationship between physical resources. Here we are not dividing the attributes according to levels, but putting all the attributes of the resource into one portrait.

The topological relationship of resources in the network layer can be divided into IP interface level, router level, and PoP (Point of Presence) level topology discovery. The resource topology can be visualized in the form of a knowledge graph. The elliptical node in the graph represents a resource entity, the circular node represents the portrait of the resource, and the relationship between the resources is represented by the edges between the elliptical nodes.

In this section, we detailed the implementation process of rowhammer PUF and evaluated the performance of the PUF under two metrics.

A. PUF Implement

We implement a rowhammer PUF on a desktop running a Linux OS based on the many-sided rowhammer tool proposed by Frigo et al [17]. We first define the challenge and response of the PUF and then give the algorithm to implement the PUF.

PUF challenge. The PUF challenge is composed of many parameters, and each parameter affects the response value of the PUF. A more stable response can be obtained by selecting appropriate challenge parameters.

- PUF address. The PUF address consists of a row base address and an address offset. The row base address is defined as the row 0 of each bank, and the starting address of the PUF is changed by changing the offset. The PUF address directly affects the PUF challenge, so the PUF address should not be the same for different PUF challenges.
- Hammering pattern. We use the many-sided (i.e., n-sided) rowhammer pattern as shown in Figure 4, where the variable is the number of aggressor rows (i.e., hammer rows) n, and its adjacent row is the victim row (i.e., PUF row). The PUF address and the hammering pattern together determine the size of the PUF. The hammer row and PUF row between different PUF challenges should not overlap.
- Data pattern. The data pattern is determined by the initial values written into the hammer row and PUF row. Due to the existence of true cell and anti cell, this parameter also has a great influence on PUF response. More bit flips can be obtained by choosing appropriate initial values for the hammer row and PUF row. For instance, initialize the hammer row to 0x00, the PUF row to 0xFF, or initialize it to 0x55 and 0xAA respectively.
- Measuring times. Measuring times includes the number of banks accessed and the number of times each hammer row is accessed. For each bank, the PUF address, hammer pattern and data pattern are all the same settings.

**PUF response.** The parameters in the PUF challenge determine the PUF response, and we use the set of cell locations where bit flips occur as the PUF response. More intuitively, an element in the PUF response set represents a location where a bit flip occurs. The format of the location is (bank, row, column), which indicates which row and column in which bank the bit flip occurs.

**PUF query.** The process of querying a rowhammer PUF is described in Algorithm 1. First initialize the hammer row and PUF row according to the PUF address, Hammering pattern, and Data pattern. The same initialization operation is required for each bank defined. Then in each bank defined, hammer each hammer row according to the hammering pattern. After that, scan the PUF row and output the location where the bit flip occurred. Finally, the set of all bit flipped locations is the PUF response.

![Fig. 4. n-sided rowhammer.](image)

---

**Algorithm 1: Process of the PUF query**

**Data:** PUF address, Hammering pattern, Data pattern, Measuring times  
**Result:** PUF response

Allocate the required memory;  
while $m < \text{number of measurements}$ do  
\hspace{1em} while $b < \text{number of banks}$ do  
\hspace{2em} Initialize hammer rows and PUF rows;  
\hspace{1em} end  
\hspace{1em} while $b < \text{number of banks}$ do  
\hspace{2em} Hammer the hammer row;  
\hspace{2em} Scan PUF rows and output bit flip locations;  
\hspace{1em} end  
end

---

**B. PUF Evaluation**

We tested the reliability and uniqueness of the PUF on two DIMMs with the same design, the same specifications, and the same production batch. The CPU model of our test computer is Intel(R) Core(TM) i7-10700 CPU @ 2.90GHz, the CPU architecture is Comet lake, and the operating system is Ubuntu 20.04. The memory is Samsung DDR4 SDRAM without ECC (Error Checking and Correcting) function, which size is 8 GB and frequency is 2932 MHz.

**Jaccard index.** One difference between DRAM PUFs and SRAM PUFs is that the response of DRAM PUFs reflects the location of the bit flipped in the memory. Therefore, the classic index of using Hamming distance to evaluate PUF characteristics is not suitable for the evaluation of DRAM PUFs. We use Jaccard index to evaluate the reliability and uniqueness of the rowhammer PUF [21]. Let $S$ denote the set of PUF responses obtained from PUF queries. The Jaccard index between two PUF responses is calculated as follows:

$$Jaccard(S_1, S_2) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}$$

which reflects the similarity between sets $S_1$ and $S_2$.

**Reliability.** Reliability measures the difference in the response of the same PUF to the same challenge in different measurements. We use index $J_{\text{intra}}(S_1, S_2)$ to represent the reliability of rowhammer PUF. Due to the measurement noise, each measurement result will not be exactly the same. Ideally, the bit flip locations contained in the two sets should be the same, so the ideal value of $J_{\text{intra}}$ is 1.

**Uniqueness.** Uniqueness measures the differences in the responses of different PUFs to the same challenge in different measurements. We use index $J_{\text{inter}}(S_1, S_2)$ to represent the uniqueness of rowhammer PUF. Ideally, the locations of bit flips contained in the two sets should not overlap, so the ideal value of $J_{\text{inter}}$ is 0.

---

**Table I**

| Parameter                | Value                        |
|--------------------------|------------------------------|
| PUF address              | row base address = 0, offset = 0 |
| Hammering pattern        | 22 sided                     |
| Data pattern             | PUF row = 0x55, hammer row = 0xAA |
| Measuring times          | 5 banks, 10 measurements     |

For each PUF query, we use the same PUF challenge and its parameter settings are shown in Table I. We fixed the PUF address to the first row in each bank and measured 5 banks. Because some DDR4 DRAMs have a TRR mitigation mechanism, some rows with bit flipping may be refreshed in the next measurement without bit flipping. Therefore, we set the number of hammer rows to 22 and measured 10 times in each PUF query. Since some cells may be true cells and some may be anti cells, we initialize the PUF row and hammer row to 0x55 and 0xAA, respectively. In this way, the states of the cells corresponding to the adjacent rows are reversed, and there are bit flipping conditions for the two possible cell situations.

Under the given challenge parameters, we performed 20 PUF queries on two DIMMs respectively. We arbitrarily select
2 query results from the 20 PUF queries to form the set $S_1$ and $S_2$, and test all possible combinations. The value of uniqueness between two DIMMs is 0, the average reliability of DIMM01 is 0.62, and the average reliability of DIMM02 is 0.63. The evaluation results are shown in Figure 5.

Figure 6 shows that the PUF reliability calculated by combining multiple PUF response queries into a set $S$ is better than the result of a single query. This means that the query results obtained by performing multiple PUF queries on the target device during the detection period can contain more physical characteristics of the device, and the identification of the device is more accurate.

**Jaccard’ index.** In the active detection process in the next section, we use rowhammer PUF to assign unique physical fingerprints to physical resources in cyberspace, and to identify and track them. According to our observations in the evaluation, the number of bit flip locations contained in different PUF queries is not the same. When the size of the two sets is very different, the value calculated by the original Jaccard index cannot accurately reflect the actual situation.

The original Jaccard index is not suitable for identifying target devices in detecting and mapping scenarios. Therefore, we modify the Jaccard index as follows:

$$Jaccard'(S_n, S_d) = \frac{|S_n \cap S_d|}{|S_n|}$$  \hspace{1cm} (2)

where $S_d$ represents a large set of PUF responses obtained from the first few queries, such as $S_d = S_1 \cup S_2 \cup S_3$. $S_n$ represents the PUF response of a new query.

The evaluation results obtained by using $Jaccard'$ index are shown in Figure 7. We arbitrarily select 3 query results from the 20 PUF response queries to form the database $S_d$, and then use the remaining single query result as $S_n$. We tested all possible combinations and got the reliability of the two test DIMMs and the uniqueness between the two DIMMs. Under the existing test data, the value of uniqueness is 0, the average reliability of DIMM01 is 0.88, and the average reliability of DIMM02 is also 0.88. The results show that the distribution
The other four devices serve as target devices to be detected. Transfer Protocol) file sharing software are deployed on it. Attack software such as Hydra and MetaSploit, and FTP (File scanning software such as Nmap and Zmap, vulnerability computer is used as the detection host, and traditional network and the gateway address was set to 192.168.171.1. A laptop experiment was carried out in a local area network environment.

V. DETECTING AND MAPPING

In this section, we carried out three sets of experiments in a laboratory environment to simulate real-world detecting and mapping scenarios. The experimental results show that our CyberRadar can accurately identify the devices in the network and obtain the portraits of the target devices.

A. Experimental Setup

The experimental architecture is shown in Figure 9. The experiment was carried out in a local area network environment and the gateway address was set to 192.168.171.1. A laptop computer is used as the detection host, and traditional network scanning software such as Nmap and Zmap, vulnerability attack software such as Hydra and MetaSploit, and FTP (File Transfer Protocol) file sharing software are deployed on it. The other four devices serve as target devices to be detected.

The portraits of target devices obtained from the detection results are shown in Table II. The portrait of the target device is composed of physical fingerprint, MAC address, IP address, port number, gateway, memory part number, operating system version, and CPU model. For each target device, enough PUF responses are collected during the detection process to form the corresponding physical fingerprint database. The content of the database is the bit flip locations corresponding to the PUF challenge we constructed.

We modified the MAC addresses, IP addresses, and open services of the four devices and then performed a new detection. The portraits of new target devices obtained are shown in Table III. The portrait of the newly detected target device is formed by combining the previously detected target device and the newly detected target device. In other words, even if the target device has replaced all its virtual attributes, we can still identify the target device through its unique physical fingerprint.

B. The First Set of Experiments

The first set of experiments used 4 computers as target devices. Among them, A1 and A2 were simulated as terminal computers used by users, A3 was simulated as a website server, and A4 was simulated as a database server to provide services for the website. By analyzing the scanned port data, we can visualize the topology between the devices as shown in Figure 10.

The portraits of target devices obtained from the detection results are shown in Table II. The portrait of the target device is composed of physical fingerprint, MAC address, IP address, port number, gateway, memory part number, operating system version, and CPU model. For each target device, enough PUF responses are collected during the detection process to form the corresponding physical fingerprint database. The content of the database is the bit flip locations corresponding to the PUF challenge we constructed.

C. The Second Set of Experiments

We modified the MAC addresses, IP addresses, and open services of the four devices and then performed a new detection. The portraits of new target devices obtained are shown in Table III.

For traditional network detection methods, these four devices will be identified as four new devices B1, B2, B3, and B4. Our goal is to identify the correspondence between the newly detected target device and the previously detected target device. In other words, even if the target device has replaced all its virtual attributes, we can still identify the target device through its unique physical fingerprint.

We match the PUF query result of the newly detected target device with the physical fingerprint database of the previously detected target device, and the results calculated according to the Jaccard index are shown in Table IV. The result shows that although device A1 changed its IP address, MAC address, and operating system version, we can still identify device A1 through physical fingerprints. Device B1 is the previous device A1, while device B2 is the previous device A4, device B3 is the previous device A2, and device B4 is still the previous device A4.
TABLE II
THE PORTRAITS OF THE TARGET DEVICES.

| Physical fingerprint | A1 | A2 | A3 | A4 |
|----------------------|----|----|----|----|
| MAC address          | Database 1 | Database 2 | Database 3 | Database 4 |
| IPv4 address         | f4:4d:30:0d:13:32 | f4:4d:30:0d:2:bf | f4:4d:30:3:2:ac:fb | f4:4d:30:2:3:91 |
| Port                 | 22:ssh | 22:ssh | 22:ssh; 44:https | 22:ssh; 3:306:mysq |
| Gateway              | 192.168.171.1 | 192.168.171.125 | 192.168.171.125 | 192.168.171.125 |
| Memory part number   | M378A1K43DB2-CVF | M378A1K43DB2-CVF | M378A1K43DB2-CVF | M378A1K43DB2-CVF |
| Operating system     | Ubuntu 20.04.3 LTS | Ubuntu 20.04.3 LTS | Ubuntu 20.04.3 LTS | Ubuntu 20.04.3 LTS |
| CPU model            | Intel(R) Core(TM) i7-10700 CPU @ 2.90GHz | Intel(R) Core(TM) i7-10700 CPU @ 2.90GHz | Intel(R) Core(TM) i7-10700 CPU @ 2.90GHz | Intel(R) Core(TM) i7-10700 CPU @ 2.90GHz |

TABLE III
THE PORTRAITS OF THE NEW DEVICES.

| Physical fingerprint | B1 | B2 | B3 | B4 |
|----------------------|----|----|----|----|
| MAC address          | PUF query 1 | PUF query 2 | PUF query 3 | PUF query 4 |
| IPv4 address         | 01:67:7a:da:2:5c | 02:6f:68:a:1:88:03 | 00:86:af:3:306:mysq | 00:86:af:3:306:mysq |
| Port                 | 22:ssh | 22:ssh | 22:ssh; 44:https | 22:ssh; 3:306:mysq |
| Gateway              | 192.168.171.1 | 192.168.171.1 | 192.168.171.1 | 192.168.171.1 |
| Memory part number   | M378A1K43DB2-CVF | M378A1K43DB2-CVF | M378A1K43DB2-CVF | M378A1K43DB2-CVF |
| Operating system     | Ubuntu 18.04.3 LTS | Ubuntu 20.04.3 LTS | Ubuntu 20.04.3 LTS | Ubuntu 20.04.3 LTS |
| CPU model            | Intel(R) Core(TM) i7-10700 CPU @ 2.90GHz | Intel(R) Core(TM) i7-10700 CPU @ 2.90GHz | Intel(R) Core(TM) i7-10700 CPU @ 2.90GHz | Intel(R) Core(TM) i7-10700 CPU @ 2.90GHz |

TABLE IV
Jaccard′ RESULTS FOR DEVICE IDENTIFICATION.

| Jaccard′ | A1 | A2 | A3 | A4 |
|----------|----|----|----|----|
| B1       | 0.96 | 0 | 0 | 0 |
| B2       | 0 | 0 | 0.94 | 0 |
| B3       | 0 | 0.95 | 0 | 0 |
| B4       | 0 | 0 | 0 | 1 |

D. The Third Set of Experiments

On the basis of the second set of experiments, we replaced two of the four devices with new ones, and the other two devices remained unchanged. Attributes such as operating system version, MAC address, IP address, and port are also unchanged and consistent with Table III.

Our goal is to identify the newly added devices. All the virtual attributes of the new devices are the same as the devices in the second set of experiments. The results calculated according to the Jaccard′ index are shown in Table IV. According to the results in Table IV, we can know that C2 and C4 are the previous devices B2 and B4.

E. One DIMM on Multiple Computers

Since we generate the unique physical fingerprint of the device based on the random characteristics of the DRAM memory on the computer. One question here is whether the physical fingerprints obtained when the same DIMM (memory module) runs on different computers will be the same, or what is the similarity. Because if the same, the same DIMM runs on two different devices at different times, the two devices will be regarded as the same device when detecting and mapping. In order to answer this question, we conducted the following experiment.

We use the same DIMM to run on three different computers A, B, and C, and get three sets of physical fingerprints. The configurations of A and B are exactly the same, and C is a computer with a different brand and configuration from A and B. The Jaccard′ index calculated according to the experimental results is shown in Table VI. The results show that even if the same DIMM runs on different devices, these devices will not be identified as the same device during detecting and mapping. This solves the problem of forging device identities by replacing DIMMs.

TABLE VI
Jaccard′ RESULTS OF THE SAME DIMM ON THREE DIFFERENT COMPUTERS.

| DIMM01 | A | B | C |
|--------|---|---|---|
| Jaccard′ | 1 | 0 | 0 |

| DIMM02 | A | B | C |
|--------|---|---|---|
| Jaccard′ | 1 | 0 | 0 |

We also calculated the average number of bit flips for the same DIMM in a PUF query on different devices A, B, and C. The results are shown in Table VII. The number of bit
flips obtained by the same DIMM on different devices is also different. The brand and configuration of device $C$ are different from $A$ and $B$. For the number of bit flips of the same DIMM, the gap between $C$ and $A, B$ is larger than the gap between $A$ and $B$.

This result seems a bit unexpected because the usual logic would think that the physical fingerprints of the same DIMM on different computers should be similar and the Jaccard value should be close to 1. We think a reasonable explanation is that the overall environment of the host computer where the DIMM is located is also an important parameter that constitutes the challenge of PUF. Although the other PUF challenge parameters are the same, the PUF challenge is not the same because the environment of the host computer is different. As a result, the PUF response is different, and the physical fingerprints obtained are different.

VI. RELATED WORK

**DRAM PUFs.** Schaller et al. [22], [23] presented the first work to design PUF using the rowhammer effect. They implemented rowhammer PUF on a PandaBoard equipped with DDR2 memory and tested the performance of PUF under different conditions. This work is groundbreaking and has brought a lot of inspiration to our work. But this work is not suitable for the detecting and mapping scenario we proposed, because we can only perform PUF queries on the target device through a remote way. Moreover, the implementation of PUF on PandaBoard does not need to consider the adddress mapping problem, and it needs to be considered when performing rowhammer on a personal computer.

For decay-based DRAM PUFs [20], [21], [24], [25], obtaining a PUF response requires disabling memory refresh for a fixed period of time. Therefore, either get the PUF response at the startup time of the device or refresh the key area of the memory to prevent the system from crashing in order to get the PUF response at runtime. This is inconsistent with the need not to affect the normal function of the detected device, and therefore it is not suitable for our detecting and mapping scenarios.

As for DRAM latency PUFs [26], [27], the PUF response is obtained by manipulating specific timing parameters to place the memory in a special state of undefined behavior. In this way, the PUF response can be obtained while the system is running, but requires the operating system to allow the DRAM timing parameters such as $rCD$ to be modified at runtime.

As far as we know, most personal computers can only modify the physical fingerprints obtained are different.

**Other Fingerprint Generation Methods.** Kohno et al. [28] proposed a remote clock skew estimation technology that uses TCP and ICPM timestamps to identify physical devices. Sanchez-Rola et al. [29] proposed a method to calculate hardware fingerprints by measuring the execution time of the CPU to run a specific instruction sequence. There is also work to extract fingerprints of hardware through GPU. Forlin et al. [30] form PUF by using the location of soft errors that occur in the GPU under unstable conditions. Li et al. [31] exploit the inherent randomness of the GPU to generate a unique, GPU-specific signature. The previous work is very meaningful, but it is difficult to apply in actual detecting and mapping scenarios.

VII. CONCLUSION

In this paper, we present a complete and detailed example of detecting and mapping. First, we propose a PUF-based detecting and mapping framework, where PUF's role is to generate a physical fingerprint that can uniquely identify the target device. Second, we gave the details of implementing rowhammer PUF on a personal computer equipped with DDR4 memory and evaluated the performance of the implemented PUF. Finally, we conducted detecting and mapping in the local area network we built. The results show that our proposed method can accurately identify and track the target device, even if the target device changes its IP address, MAC address, or even the operating system.

**Limitation and Future Work.** Our PUF-based detecting and mapping framework currently uses rowhammer PUF to generate physical fingerprints. The current rowhammer technology cannot guarantee bit flipping on all DDR4 memory, especially memory with ECC function. There are also many protection mechanisms such as TRR (Target Row Refresh) that have been proposed to prevent bit flipping. But at the same time, new rowhammer technologies are constantly being developed, and most of the devices deployed on the Internet are not up-to-date and will still be affected by rowhammer. Therefore, we plan to develop new device fingerprint generation methods suitable for detecting and mapping, and carry out real-world cyberspace detecting and mapping in the future. And we plan to develop automated resource portrait building tools and topology visualization tools.

REFERENCES

[1] G. Li, C. Yanan, S. Majing, S. Yanmin, Z. Yujia, Z. Peng, and Z. Chuan, “Cyberspace resources surveying and mapping: the concepts and technologies,” Journal of Cyber security, vol. 3, no. 4, pp. 1–14, 2018.

[2] R. Xu, Z. Zhang, Z. Rao, J. Chen, M. Li, F. Liu, and S. Pan, “Cyberspace surveying and mapping: Hierarchical model and resource formalization,” in IEEE INFOCOM 2019–IEEE Conference on Computer Communications Workshops. IEEE, 2019, pp. 68–72.

[3] Y. Hou, X. Chen, Y. Hao, Z. Shi, and S. Yang, “Survey of cyberspace resources scanning and analyzing,” in International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing. Springer, 2020, pp. 279–291.

[4] Nmap: a free network mapping and security scanning tool, [http://nmap.org](http://nmap.org)

[5] Z. Durumeric, E. Wustrow, and J. A. Halderman, “Zmap: Fast internet-wide scanning and its security applications,” in 22nd USENIX Security Symposium (USENIX Security 13), 2013, pp. 605–620.
A. Schaller, W. Xiong, N. A. Anagnostopoulos, M. U. Saleem, S. Gabmeyer, S. Sutar, A. Raha, and V. Raghunathan, “D-puf: An intrinsically reconstructible and secure dram puf,” in 2016 IEEE International Symposium on Hardware Oriented Security and Privacy (HOST). IEEE, 2016, pp. 161–166.

A. Kwon, D. Genkin, D. Gruss, and Y. Yarom, “Rambleed: Reading bits in memory without accessing them,” in 25th USENIX security symposium (USENIX security 16), 2016, pp. 565–581.

M. Wang, Z. Zhang, Y. Cheng, and S. Nepal, “DRAMBind: A knowledge-assisted tool to uncover dram address mapping,” in 2020 57th ACM/IEEE Design Automation Conference (DAC). IEEE, 2020, pp. 1–6.

D. Gruss, M. Lipp, M. Schwarz, D. Genkin, J. Juffinger, S. O’Connell, W. Schoechl, and Y. Yarom, “Another flip in the wall of rowhammer defenses,” in 2018 IEEE Symposium on Security and Privacy (SP). IEEE, 2018, pp. 245–261.

P. Frigo, E. Vannace, H. Hassan, V. Van Der Veen, O. Mutlu, C. Giuffrida, H. Bos, and K. Razavi, “Trespass: Exploiting the many sides of target row refresh,” in 2020 IEEE Symposium on Security and Privacy (SP). IEEE, 2020, pp. 747–762.

V. Van Der Veen, Y. Fratantonio, M. Lindorfer, D. Gruss, C. Maurice, G. Vigna, H. Bos, K. Razavi, and C. Giuffrida, “Dranner: Deterministic rowhammer attacks on mobile platforms,” in Proceedings of the 2016 ACM SIGSAC conference on computer and communications security, 2016, pp. 1675–1689.

S. S. Zaliva, A. A. Ivanik, and C.-H. Chang, “Reliable and modeling attack resistant authentication of arbiter puf in fpga implementation with trinary quadruple response,” IEEE Transactions on Information Forensics and Security, vol. 14, no. 4, pp. 1109–1123, 2018.

F. Tehranipoor, N. Karimian, W. Yan, and J. A. Chandy, “Drum-based intrinsic physically unclonable functions for system-level security and authentication,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 25, no. 3, pp. 1085–1097, 2016.

W. Xiong, A. Schaller, N. A. Anagnostopoulos, M. U. Saleem, S. Gabmeyer, S. Katzenbeisser, and J. Szefer, “Run-time accessible DRAM PUFs in commodity devices,” in International Conference on Cryptographic Hardware and Embedded Systems. Springer, 2016, pp. 432–453.

A. Schaller, W. Xiong, N. A. Anagnostopoulos, M. U. Saleem, S. Gabmeyer, S. Katzenbeisser, and J. Szefer, “Intrinsic rowhammer PUFs: Leveraging the rowhammer effect for improved security,” in 2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST). IEEE, 2017, pp. 1–7.

N. A. Anagnostopoulos, T. Arul, Y. Fan, C. Hatzfeld, A. Schaller, W. Xiong, M. Jain, M. U. Saleem, J. Lotichius, S. Gabmeyer et al., “Intrinsic run-time row hammer PUFs: Leveraging the row hammer effect for run-time cryptography and improved security,” Cryptography, vol. 2, no. 3, p. 13, 2018.

S. Sutar, A. Raha, and V. Raghunathan, “D-puf: An intrinsically recon-figurable dram puf for device authentication in embedded systems,” in 2016 International Conference on Compilers, Architectures, and Synthesis of Embedded Systems (CASES). IEEE, 2016, pp. 1–10.

A. Schaller, W. Xiong, N. A. Anagnostopoulos, M. U. Saleem, S. Gabmeyer, B. Skorit, S. Katzenbeisser, and J. Szefer, “Decay-based dram pufs in commodity devices,” IEEE Transactions on Dependable and Secure Computing, vol. 16, no. 3, pp. 462–475, 2018.

J. S. Kim, M. Patel, H. Hassan, and O. Mutlu, “The dram latency puf: Quickly evaluating physical unclonable functions by exploiting the latency-reliability tradeoff in modern commodity dram devices,” in 2018 IEEE International Symposium on High Performance Computer Architecture (HPCA). IEEE, 2018, pp. 194–207.

J. Miskelly and M. O’Neill, “Fast dram pufs on commodity devices,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 39, no. 11, pp. 3566–3576, 2020.

T. Kohno, A. Broido, and K. C. Claffy, “Remote physical device fingerprinting,” IEEE Transactions on Dependable and Secure Computing, vol. 2, no. 2, pp. 93–108, 2005.

I. Sanchez-Rola, I. Santos, and D. Balzarotti, “Clock around the clock: Time-based device fingerprinting,” in Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, 2018, pp. 1502–1514.

B. Forlin, R. Husemann, L. Carro, C. Reinbrecht, S. Handziou, and M. Taouil, “G-puf: An intrinsic puf based on gpu error signatures,” in 2020 IEEE European Test Symposium (ETS). IEEE, 2020, pp. 1–2.

F. Li, X. Fu, and B. Luo, “Poster: A hardware fingerprint using gpu core frequency variations,” in Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, 2015, pp. 1650–1652.