Decay-Based DRAM PUFs in Commodity Devices

André Schaller, Wenjie Xiong, Nikolaos Athanasios Anagnostopoulos, Muhammad Umair Saleem, Sebastian Gabmeyer, Boris Škorić, Stefan Katzenbeisser, and Jakub Szefer

Abstract—A Physically Unclonable Function (PUF) is a unique and stable physical characteristic of a piece of hardware, which emerges due to variations in the hardware fabrication processes. Prior works have demonstrated that PUFs are a promising cryptographic primitive that can enable secure key storage, hardware-based device authentication and identification. So far, most PUF constructions have required an addition of new hardware or an FPGA implementation for their operation. Recently, intrinsic PUFs, which can be found in commodity devices, have been investigated. Unfortunately, most of them suffer from the drawback that they can only be accessed at boot time. This paper focuses on a new class of run-time accessible, decay-based, intrinsic DRAM PUFs in commercial off-the-shelf systems, which requires no additional hardware or FPGAs. In order to enable secure key storage using DRAM PUFs, this work presents a new Helper Data System (HDS) specifically tailored to the properties of the decay process inherent to DRAM cells. The decay-based DRAM PUF and the new HDS are evaluated on commodity off-the-shelf devices to demonstrate their practicality. Furthermore, a novel lightweight protocol is presented that allows for mutual authentication.

Index Terms—Physically unclonable functions, helper data schemes, device authentication

1 Introduction

Miniaturization and cost reduction of processors and System-on-Chip designs have enabled the creation of almost ubiquitous smart devices, from smart thermostats and appliances, to smart phones and embedded car entertainment systems. With the proliferation of smart devices, new security vulnerabilities are constantly discovered, e.g., [1], [2], [3], [4]. One major concern is that these devices often lack implementation of sufficient security mechanisms [5], [6]. The lack of secure hardware components, as well as constraints on memory and computational power concern the security of these devices. Establishing means of providing robust device authentication and identification mechanisms, and means to store long-term cryptographic keys in a secure manner that minimizes the chances of their illegitimate extraction or access are particularly demanding.

A common approach to device identification is to embed cryptographic keys in each device by burning them in at manufacturing time. However, this solution comes with potential pitfalls, such as increased production complexity as well as rather limited protection against key extraction attempts [7]. As an alternative, researchers have proposed Physically Unclonable Functions (PUFs). PUFs leverage the unique behavior of a device due to manufacturing variations as a hardware-based fingerprint. Since the exact variations present in one device are extremely difficult to replicate in another device, even by the manufacturer, PUFs cannot be easily cloned. Moreover, the variations are stable, robust, and unique to each device. Hence, PUFs have been proposed as cryptographic building blocks for security primitives and protocols, such as authentication and identification [8], [9], [10], hardware-software binding [11], [12], [13], [14], [15], remote attestation [16], [17], and secret key storage [18], [19]. So far, most types of PUFs in digital electronic systems (such as arbiter PUFs [8], [20]) require the addition of dedicated circuits to the device and thus increase manufacturing costs and hardware complexity. Consequently, there is great interest in so-called intrinsic PUFs [11], which are PUFs that rely on hardware components that are inherent to virtually any device. Two examples are Static Random-Access Memory (SRAM) based PUFs, and Dynamic Random Access Memory (DRAM) based PUFs. DRAM PUFs are focus of this work.

Intrinsic PUFs are an attractive, low-cost security anchor, as they provide PUF instances within standard hardware that can be found in commercial off-the-shelf devices [21], [22], and thus do not require any hardware modifications. The most prominent example of intrinsic PUFs are those based on the aforementioned SRAM modules [13], [14], [23], [24], [25], which draw their characteristics from the startup values of bi-stable SRAM cells. SRAM PUFs are known to have good PUF characteristics [26]. However, PUF measurements must be extracted during a very early boot stage (before the SRAM is written to). Consequently,

- *A. Schaller, N.A. Anagnostopoulos, M.U. Saleem, S. Gabmeyer, and S. Katzenbeisser are with Technische Universität Darmstadt, Darmstadt, Hessen 64289, Germany. E-mail: {schaller, anagnostopoulos, gabmeyer, katzenbeisser}@seceng.informatik.tu-darmstadt.de, muhammadumair.saleem@stud.tu-darmstadt.de.*
- *W. Xiong and J. Szefer are with Yale University, New Haven, CT 06520. E-mail: {wenjie.xiong, jakub.szefer}@yale.edu.*
- *B. Škorić is with the Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven 5612 AZ, Netherlands. E-mail: b.skoric@tue.nl.*

Manuscript received 4 May 2017; revised 18 Jan. 2018; accepted 27 Mar. 2018. Date of publication 6 Apr. 2018; date of current version 10 May 2019. (Corresponding author: André Schaller.)
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TDSC.2018.2822298

the derived key can only be used at this time, or must be saved in some external memory, which may itself cause new security problems by exposing the key to malicious extraction attempts. Recently, a new error-based SRAM PUF, which can be accessed at run-time, was proposed [27]. However, to query the PUF, the supply voltage needs to be lowered to induce errors in SRAM cells, requiring special hardware in the processor.

Meanwhile, Dynamic Random-Access Memory PUFs have been proposed recently [28]. One approach to extract unique DRAM behavior induced by manufacturing variations relies on startup tendencies of DRAM cells [29], [30], [31]. Another approach to extract DRAM PUFs is to leverage the unique decay characteristics of DRAM cells and exploit the fact that charges of individual DRAM cells, if not refreshed, decay over time in a unique manner [31], [32]. PUF responses[1] can be generated by initializing DRAM cells with a specific value, disabling DRAM refresh cycles, and letting the cells decay for a defined *decay time*. As a result of this decay, a DRAM chip exhibits bit flips at various locations within the memory. The unique distribution of locations of the bit flips can be used as a PUF response. Prior to our recent work [33], state of the art required custom hardware or FPGA-based platforms [28], [29], [30], [31], [32] in order to modify the DRAM refresh mechanism such that DRAM PUF extraction is possible.

1.1 Contributions to Decay-Based DRAM PUF Design

Our research is the first to deal with decay-based DRAM PUFs in commodity devices. This paper is an expanded version of our conference publication [33], which introduced intrinsic DRAM PUFs. This work extends the prior paper with the following contributions:

- Enhanced evaluation of decay-based DRAM PUFs compared to [33], with measurements using decay times that are up $6\times$ faster, covering larger memory regions and wider temperature ranges on 9 devices covering two different kinds of commodity off-the-shelf platforms; extended time stability measurements spanning 16 months in total.
- Design of a novel, lightweight, and compact Helper Data System, specifically tailored towards decay-based DRAM PUFs, enabling efficient key storage.
- Development of a lightweight authentication protocol that achieves mutual authentication.

1.2 Related Work on PUFs

A Physically Unclonable Function (PUF) has shown to be a promising cryptographic primitive. Different PUF implementations have been proposed, e.g., delay-based PUFs and memory-based PUFs. Delay-based PUFs often require dedicated circuits, such as arbiters and ring oscillators [8], [20]. In contrast, memory-based *intrinsic* PUFs leverage variations in storage cells already present on the computing devices, such as SRAM [23], [24], [25], [26], Flash memory [34], [35], and DRAM. The earliest approach to exploit manufacturing variations of DRAM cells for identification and random number generation was reported in [28], [36], where a DRAM chip is designed to generate fingerprints to mitigate hardware counterfeiting. In subsequent work, through a memory controller synthesized in an FPGA, Keller et al. [32] proposed to use the decay of external DDR3 modules for extracting random bits and unique identifiers. Lui et al. [37] evaluated the uniqueness, robustness, and min-entropy of external DRAM modules using an FPGA setup, and proposed a secure key storage scheme. Hashemian et al. [38] designed a circuit exploiting the varying reliability during write cycles of DRAM cells and presented an authentication scheme based on such generated signatures. Rehmati et al. [39] made use of the error patten in approximate DRAM as a system fingerprint. Tehranipoor et al. [29], [30] exploit startup values of DRAM cells to extract a device signature. Sutar et al. [31] evaluated the DRAM PUF with an FPGA setup and proposed an authentication scheme with reduced authentication time by reconfiguring the DRAM for different decay times.

Unlike this work, all previous research required dedicated circuits to be designed or FPGAs to be used. To the best of our knowledge, our work in [33] and the extended work presented in this paper are the first contributions that focus on intrinsic decay-based DRAM PUF instances in commodity devices, accessible at run-time. We also provide a system-level solution for querying the PUF while a Linux OS is running on the same hardware and actively using the DRAM chip wherein the PUF is located.

1.3 Decay-Based DRAM PUFs in Commodity Devices

Our research shows that a run-time accessible PUF can be constructed from the decay behavior of DRAM that is part of unmodified commodity devices, including the PandaBoard and the Intel Galileo platforms. Two approaches are evaluated: (i) accessing the PUF at device startup using a customized firmware, and (ii) querying the PUF using a kernel module at run-time.

Through extensive experiments on multiple instances of two types of commodity devices, we show that DRAM PUFs exhibit robustness, uniqueness, and in particular allow usage of the decay time as part of the PUF challenge. Especially, evaluations of shorter decay times, which are up to 6 times faster, and larger memory regions of $16\,\mathrm{MB}$ give extensive insights into the practicality of decay-based DRAM PUFs.

In [33] we introduced new metrics for evaluating DRAM PUFs, based on the Jaccard index, and showed that they are, in contrast to classic Hamming distance-based metrics, better suited regarding the particular properties of decay-based DRAM PUFs. The inter and intra Jaccard index were used to compare uniqueness and robustness of DRAM PUFs. We further estimated the entropy contained in the PUF measurements by means of the Shannon entropy.

In this work, we present a novel Helper Data System (HDS) tailored to the properties of decay-based DRAM PUFs, which incorporates an enrollment phase that uses a few quick measurements to locate fast-decaying cells, and one long-timescale measurement to locate slowly decaying cells. Only exceptionally fast and slow cells are selected as input to the HDS. Our selection method solves the problem of large biases towards '0' or '1' in the PUF measurements that occur due to the vast discrepancy between the number

1. In the following we will use the terms PUF *measurement* and PUF *response* interchangeably.

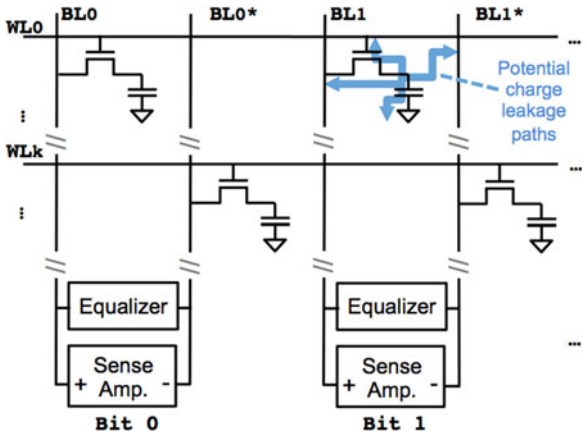


Fig. 1. A single DRAM cell consists of a capacitor and a transistor, connected to a word-line (WL) and a bit-line (BL or BL*); arrows indicate leakage paths for dissipation of charges that lead to PUF behavior.

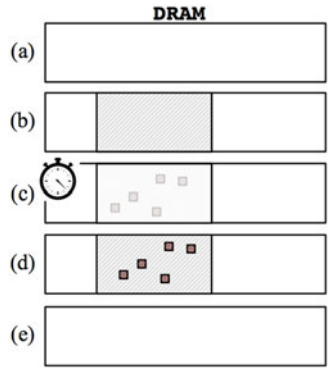


Fig. 2. Five steps required for run-time access of a DRAM PUF. Only during steps (b)—(d) the memory associated with the PUF is not usable for any other processes.

of fast cells and the much higher amount of slow cells. We further optimize the proposed HDS towards the properties of the devices under test, to only require a single enrollment measurement. The described HDS is very simple to implement and considerably compact, as the relevant helper data require only minimal memory space to be stored. The HDS is experimentally validated to work even for highly biased PUF measurements.

We also present a new lightweight, mutual PUF-based authentication protocol. It can be used in resource-constrained devices which implement DRAM, but do not possess the processing power to run the costly cryptographic algorithms, such as many of the smart devices found today.

1.4 Outline

The remainder of the paper is organized as follows. Section 2 presents background on DRAM, introduces our decay-based DRAM PUF and discusses security assumptions. Implementation details of the DRAM PUF on two evaluation platforms are given in Section 3. Section 4 contains our evaluation of DRAM PUFs characteristics extracted from multiple devices. In Section 5 we present a Helper Data System suitable for key storage in DRAM PUFs. In Section 6 we present a novel lightweight authentication protocol that uses the DRAM PUF. We finally conclude our work in Section 7.

2 DRAM PUFs in Commodity Devices

Fig. 1 shows an array of typical DRAM cells. A single DRAM cell stores a charge in a capacitor and can be accessed through a transistor. DRAM cells are grouped into arrays, where each row of the array is connected to a horizontal word-line and DRAM cells in the same column are connected to the same bit-line. All bit-lines are coupled to equalizers and sense-amplifiers that amplify voltages on bit-lines to levels such that they can be interpreted as logical zeros or ones. In order to access a row, all bit-lines will be *precharged* to half the supply voltage $V_{DD}/2$. Subsequently the connected word-line is *enabled*, activating every transistor in that line and allowing charges form the capacitors to flow to their associated bit-lines. The sense amplifier then drives the bit-line to V_{DD} or $0V$, depending on the charge that was stored on the capacitor. The amplifiers are usually shared by two bit-lines [40], of which only one can be accessed at the same time. This structure makes the two bit-lines complementary, which results in two kinds of cells: true-cells and anti-cells. True-cells store the value '1' as V_{DD} and '0' as 0V on the capacitor, while anti-cells store the value '0' as V_{DD} and '1' as 0V.

DRAM cells require periodic refresh of the stored charges, as otherwise the capacitors lose their charge over time, which is referred to as DRAM cell *decay* or *leakage*. The hardware memory controller takes care of periodic refresh, whose interval is defined by the vendor, and is usually 32 ms or 64 ms. Without this periodic refresh, the logical value of true cells decay to '0', while anti-cells decay to '1'. Because of the manufacturing variations among DRAM cells, some cells decay faster than others. The unique decay characteristics of individual DRAM cells can be exploited for a decay-based DRAM PUF, as our research shows.

2.1 Decay-Based PUFs in DRAM

The process of exploiting the unique decay behavior of DRAM cells in order to extract a PUF measurement is summarized in Fig. 2. The starting point (a) comprises the DRAM module being configured for ordinary use, where the memory controller periodically refreshes all of the cells' content. In a first step (b), the PUF memory region, defined by starting address ($addr$) and size ($size$), is reserved such that it does not contain any user-space or operating system (OS) programs. This region is depicted as a shaded gray rectangle in the figure. The reservation can be implemented using memory ballooning introduced later in Section 2.2. Furthermore, the refresh for the PUF region is disabled and the initialization value (iv) is written to the region. Next, (c) for a given decay time (t), the memory region containing the PUF is not accessed to let the cells decay. After the decay time has expired, (d) the memory content is read in order to extract the PUF measurement. At the end, (e) the normal operating condition of the memory is restored and the memory region is made available to the OS again.

Memory regions within a DRAM module that are used for obtaining PUF measurements are called *logical DRAM PUFs*. For a particular DRAM, each logical PUF is determined by: (i) $addr$, the starting address of the logical PUF, and (ii) $size$, its size, as discussed above. A typical DRAM memory module can then be divided into thousands or more logical PUFs.

Two additional parameters are needed to define a DRAM PUF challenge, the initialization value (iv) that will be written to the DRAM PUF cells before any decay process starts, and the desired decay time (t). After the decay time has expired, enough charge has leaked from some cells such that their stored logical bits have flipped. As the positions of the flipped bits are unique for individual DRAM regions, the "pattern" of decayed bits, also referred to as flipped bits, for a given decay time t serves as the PUF response.

In order to derive a cryptographic key from the PUF response using a minimum number of DRAM cells, the entropy within a logical DRAM PUF response needs to be maximized. The value stored in a DRAM cell before it decays, iv, plays an important role, as some DRAM cells decay to '0' and some to '1'. Thus, for example, if a cell decays to '0', but its initialization value is set to '0', the decay effect cannot be observed. If the physical layout of the DRAM module is known (i.e., the distribution of true-cells and anti-cells, and hence the individual decay directions), it is possible to construct an initialization value that maximizes the number of observable bit flips in the PUF response. However, the physical layout is rarely known. Furthermore, the optimal initialization value would need to be part of the challenge, or it would have to be stored on the device. In our evaluation, we use a fixed initialization value iv of '0' for all cells within the memory being used as PUF. Thus, the entropy of our measurements can further be improved if the initialization value is varied so that each cell is initialized with a logical value that corresponds to a state, where charge is stored on the cells' capacitor (i.e., '1' for true-cells, and '0' for anti-cells).

Overall, the challenge of a DRAM PUF consists of PUF_{id} and t, where PUF_{id} denotes the logical PUF instance ($addr$ and $size$) and t denotes the decay time after which the memory content is read. In our experiments we fixed the value of iv, hence we do not specify the parameter explicitly.

Although SRAM and DRAM PUFs are both considered weak PUFs [41], the DRAM PUF presented in this paper offers multiple challenges due to the ability to vary decay times t. Given two PUF measurements m_x and m_{x+1} from the same logical PUF_{id}, taken at decay times t_x and t_{x+1} ($t_{x+1} \geq t_x$), both m_{x+1} and m_x can serve as PUF responses. We denote the set of addresses of decayed (i.e., "flipped") DRAM cells at decay time t as $s(t)$. With increasing decay time t, the number of DRAM cells flipping is monotonically increasing. In particular, m_{x+1} consists of a number of *newly* flipped bits as well as the majority[2] of bits that already flipped in m_x. In general, if $t_x \leq t_{x+1}$ and $addr_x = addr_{x+1}$, $size_x = size_{x+1}$, we observe $s(t_x) \subseteq s(t_{x+1})$, up to noise. However, note that it is not possible to measure responses for several decay times $t_0, t_1, \ldots, t_n$ at once. In particular, reading the PUF response at one decay time will cause the memory to be refreshed (the cells are re-charged, as data is read from DRAM cells into row buffers). Querying a PUF response with a different decay time thus requires to restart the experiment.

2. Due to noise, the set of flipping cells for a fixed time t_x will not be completely stable. Nevertheless, our experiments in Section 4 show very low amounts of noise.

2.2 Run-Time DRAM PUF Access

Deactivating DRAM refresh for PUF access during device operation is a non-trivial task: when DRAM refresh cycles are disabled, critical data (such as data belonging to the OS or user-space programs) will start to decay and the system will crash. In our experiments, the Intel Galileo board running Yocto Linux crashes about one minute after DRAM refresh is disabled. Therefore, we present a customized solution, which allows us to refresh critical code but leaves memory regions dedicated to PUF usage untouched. This solution is based on two techniques dubbed *selective DRAM refresh* and *memory ballooning*. The former allows for selectively refreshing memory regions occupied by the OS and other critical applications so that they run normally and do not crash. Memory ballooning, on the other hand, safely reserves the memory region that corresponds to a logical PUF without corrupting critical data and also protects the memory region from accesses by OS and user-space programs, to allow the DRAM cells to decay without being disturbed during the PUF measurements.

Selective DRAM Refresh. On some devices, such as the PandaBoard, DRAM consists of several physical modules or logical segments, where the refresh of each module/segment can be controlled individually. In this case, the PUF can be allocated in a different memory segment from the OS and user-space programs. When challenging the PUF, only the refresh of the segment holding the PUF is deactivated, while the other segments remain functional.

On other devices, e.g., the Intel Galileo, the refresh rate can only be controlled at the granularity of the entire DRAM.[3] Refresh at segment granularity is not possible. However, memory rows can be refreshed implicitly once they are accessed due to a read or a write operation. When a word line is selected because of a memory access, the sense amplifier drives the bit-lines to either the full supply voltage V_{DD} or back down to $0\mathrm{V}$, depending on the value that was in the cell. In this way, the capacitor charge is restored to the value it had before the charge started to leak. Using the above principle, even if refresh of the whole memory is disabled, selective memory rows can be refreshed by issuing a read to a word within each of the selected memory rows periodically. This functionality can be implemented in a kernel module by reading a word within each memory row to be refreshed (Section 3).

Ballooning System Memory. To query a chosen logical PUF, the DRAM portion given by $addr$ and $size$ is overwritten by the respective initialization value (iv) and refresh is deactivated. To prohibit applications from accessing the PUF and thus implicitly refreshing them, we use memory ballooning concepts developed for virtual machines [42]. Memory ballooning is a mechanism for reserving a portion of the memory so as to prevent the memory region from being used by the kernel or any application. This approach allows to specify the physical address ($addr$) and size ($size$) of the PUF memory region that will be reserved. Once PUF memory is "ballooned", DRAM refresh can be disabled and selective

3. Although the test boards do have multiple DRAM modules, DRAM refresh cannot be disabled individually. In particular, on the Galileo board, one DRAM chip is used to store the most significant 8 bits of every 16 bits, while the other chip is used to store the least significant 8 bits. Disabling refresh on a single chip is not possible, as half of each memory word would be lost.

refresh enabled for the non-PUF memory region. If access to the PUF is no longer required, the balloon can be deflated and the memory restored to normal use.

2.3 Security Assumptions

DRAM PUFs differ from classic memory-based PUFs, as they can be evaluated during run-time. An attacker, who wants to evaluate the PUF needs to disable DRAM refresh. This task requires writing to hardware registers, which can only be performed by the kernel. An attacker thus requires root privileges. Furthermore, accessing the memory dedicated to the PUF itself is restricted to the kernel as well. Thus, a crucial security assumption is that firmware and operating system are trusted and an attacker does not gain root privileges.

An attacker may try to change the ambient temperature in order to influence the bit flip characteristics. Nevertheless, a legitimate user can compensate the temperature effect by adjusting the decay time, as discussed in Section 4. The attacker could also try to adapt the "rowhammering" approach presented in [43], i.e., inducing random bit flips into DRAM cells by repeatedly accessing adjacent rows. However, the attacker would not succeed, as DRAM PUFs allocate a continuous chunk of memory. Rowhammering would only apply at the borders of the PUF area. At the same time, the rowhammer effect can be leveraged to obtain DRAM PUFs with higher entropy [44].

Although voltage variations can affect PUF behaviour, as shown in [28], [38], changing the voltage supply of DRAM on commodity hardware, without affecting the supply of other components, such as the MCU, is not trivial, even when it is possible. We therefore consider the effects of voltage variations to be out of scope. Invasive attacks are also considered out of scope.

Finally, we consider aging as a factor that could affect the stability of DRAM [29], [30], [45] and therefore could be used in attacks. Here, we examine naturally occurring aging, and not accelerated aging, as on commodity hardware it is not trivial to manipulate aging effects only for the DRAM unit and not for the other components of the test platform.

3 Implementation & Performance

We implemented and tested our DRAM PUF construction on two popular platforms, the PandaBoard ES Revision B3 and the Intel Galileo Gen 2. The PandaBoard houses a TI OMAP 4460 System-on-Chip (SoC) module that implements 1 GB of DDR2 memory from ELPIDA in a Package-on-Package (PoP) configuration, which operates at 1.2 V. The Intel Galileo is equipped with an Intel Quark SoC X1000 SoC and two 128 MB DDR3 from Micron, operating at 1.5V. The two physical DRAM modules are accessed in parallel and located on the same PCB as the processor.

We implemented two different approaches to query the PUF. The first approach uses a modified firmware in order to obtain PUF measurements during the boot phase. Second, we implemented a kernel module-based solution that enables PUF queries during run-time of a Linux operating system. The firmware solution can be implemented in a straight-forward fashion and was used to take most of the measurements from the Intel Galileo. The kernel module-based solution was used for obtaining measurements on the PandaBoard platform and for gathering temperature stability measurements on both platforms. The kernel module thus also serves as a general proof-of-concept of the run-time accessibility of the proposed DRAM PUF. We present implementation details of both approaches in the following.

3.1 Firmware-Based PUF Access

The firmware is the first code to be executed upon device start. During the DRAM initialization phase, the firmware itself does not require the use of DRAM, as it is executed from on-chip SRAM. This makes it ideal for accessing PUF.

In the case of the Galileo platform, we modified the Quark EDKII firmware. Code that measures the PUF was inserted just before DRAM refresh, comprising the following steps: writing the initial value (iv) to the specific logical PUF (as defined by $addr$ and $size$), waiting for the decay time t to elapse, and then reading back the PUF response via the console. After the PUF response is retrieved, normal firmware execution and eventual boot of the OS can resume. The firmware patch consists of about 60 lines of C code. The majority of the code implements initialization of the PUF parameters and accessing the PUF memory region. The PUF response is read and printed to the console for later analysis.

On the PandaBoard, the implementation is similar: the DRAM region corresponding to the PUF is initialized, the auto-refresh of the memory controller is disabled, and after decay time t, the memory content is sent over UART to a workstation. Our firmware patch for the PandaBoard consists of about 50 lines of C code.

3.2 Linux Kernel Module-Based PUF Access

In order to be able to access the DRAM PUF during run-time, we implemented a kernel module for each platform, which can be inserted at run-time. The kernel module is designed to work in three phases: (1) Upon loading, the kernel module overwrites the contents of the DRAM cells in the desired logical PUF region with iv. (2) The kernel module then modifies the memory controller via writes to configuration registers to disable DRAM refresh, while memory locations occupied by the OS and applications are selectively refreshed, as explained in Section 2.2. (3) After the decay time of t seconds has elapsed, memory refresh is enabled again and the PUF response is read out.

On the PandaBoard, DRAM can be accessed using two individual external memory interfaces (EMIF), with each EMIF covering 512 MB. In our implementation, memory interfaced by the first EMIF can be used by the kernel and user space applications, while memory covered by the second EMIF can be used exclusively as DRAM PUF. In order to implement this configuration, the interleaving mechanism of the PandaBoard that alternately maps subsequent logical addresses to physical addresses from both modules must first be disabled within the bootloader. Next, measurements can be obtained by turning off the refresh rate of the module that implements the logical PUFs and reading the memory contents after the decay time t, while the kernel and user space applications remain functional on the other DRAM module. The kernel module takes about 100 lines of C code in total.

On the Intel Galileo, refresh of the whole DRAM has to be disabled as it is not possible to control refresh at a smaller granularity than a DRAM module. Consequently, the kernel module must selectively refresh memory used by the kernel

TABLE 1
Time Needed to Perform Memory Reads, to Selectively Refresh Varying Sizes of Memory Regions on the (Single-Core, Single-Threaded) Intel Galileo Board with DDR3 Memory

system memory	selective refresh time	%CPU time (64 ms refresh)	%CPU time (200 ms refresh)
32 MB	7.6 ms	12%	4%
64 MB	12.1 ms	19%	6%
128 MB	21.2 ms	33%	10%

and applications. The kernel module schedules selective refresh tasks[4] every N ms, where N is the desired refresh rate. For selective refresh, the module loops over all memory addresses that need to be refreshed, issuing a read to a memory word in every DRAM row. The kernel module takes about 300 lines of C code in total.

During a PUF query, the OS and other applications can operate normally, but some CPU resources must be spent on selective memory refresh. If the size of the memory region is too large, the CPU will spend the majority of its time refreshing the defined memory area, leaving little resources to user space applications. Furthermore, if the time required to refresh the whole memory region is longer than the required refresh period, critical portions of code and data may have decayed before they can be refreshed by the kernel module, causing a system crash.

Table 1 shows the time required to perform selective refresh of system memory regions of various sizes, ranging from 32 MB up to 128 MB. We see that selective refresh takes between 7.6 ms and 21.2 ms for a single run. The last two columns in Table 1 show the CPU time spent on selective refresh, assuming 64 ms and 200 ms refresh rates. For an active memory size of 128 MB, the system will spend 33 percent of CPU time on selective refresh when a target refresh period of 64 ms is selected. However, at room temperature, the 64 ms refresh period, picked by most vendors, is very conservative, and our experiments suggest that even with a refresh rate of 200 ms DRAM content remains stable. Previous work on DRAM retention time supports our results [46]. Thus, depending on the operating conditions and required stability guarantees, the selective refresh period can be increased, allowing larger DRAM to be refreshed, or leaving more CPU resources for computation. In our setup, we were able to reduce the memory footprint of Yocto Linux, commonly used on Intel Quark devices, down to 32 MB without any special modifications.[5] At 32 MB, only 7.6 ms are needed for selective refresh every 64 ms, making more than 87 percent CPU time available for other applications. These numbers demonstrate that selective refresh is viable for realistic code sizes.

4 Evaluation of Decay-Based DRAM PUFs

We measured DRAM PUF instances on the Intel Galileo and PandaBoard, as described in Section 3. We performed measurements using 4 different PandaBoards and 5 Intel Galileo devices. Furthermore, given the large amount of memory present, we measured two logical PUFs on each device, resulting in eight different logical PUFs for the PandaBoard as well as ten logical PUFs for the Intel Galileo. Each logical PUF was measured at different decay times, with 50 measurements each. We used two groups of configurations. One group resembles the experimental setup presented in [33] with decay times $\mathcal{T}_2 = \{120\,\text{s}, 180\,\text{s}, 240\,\text{s}, 300\,\text{s}, 360\,\text{s}\}$ and a smaller PUF size of 32 KB. The other group uses new decay times $\mathcal{T}_1 = \{10\,\text{s}, 20\,\text{s}, 30\,\text{s}, 40\,\text{s}, 50\,\text{s}, 60\,\text{s}\}$, which are up to six times faster and further covers a larger PUF size of 16 MB. Based on these measurements we evaluated robustness, uniqueness, randomness, time and temperature dependency, as well as stability of the DRAM PUFs. In order to present a realistic scenario, we tested our devices under conditions that naturally vary over time, in order to resemble ambient properties during real-world usage.

The characteristics of the DRAM PUFs are different compared to SRAM PUFs. Rather than being considered as an array of bits, a DRAM PUF response consists of the positions of *decayed* cells in a memory region. Thus, the standard metrics commonly used to evaluate memory-based PUFs (usually fractional Hamming distances) are not suitable for DRAM PUFs. This is particularly noticeable when evaluating uniqueness. In SRAM PUFs the fractional Hamming distance between the startup arrays of two different PUFs is large, whereas for DRAM PUFs the distance is small, even if PUFs are highly unique. This effect is caused by the fact that the majority of DRAM cells does not decay within typical timescales of PUF challenges.

We propose new robustness and uniqueness metrics that ignore the 'uninteresting' majority of cells, i.e., those cells that did not decay. We use these metrics to evaluate multiple instances of DRAM decay-based PUFs as shown in Table 2. Our metrics are based on the Jaccard index [47], which is a well known metric to quantify the similarity of two sets of different size. It results in a value of zero if the sets share no common elements and a value of one if the sets are identical. The Jaccard index of two sets $\mathcal{A}, \mathcal{B}$ is defined as

$$J(\mathcal{A}, \mathcal{B}) \stackrel{\text{def}}{=} \frac{|\mathcal{A} \cap \mathcal{B}|}{|\mathcal{A} \cup \mathcal{B}|}. \tag{1}$$

Uniqueness. Consider two DRAM PUFs, PUF_{id_1} and PUF_{id_2}, which are given time t to decay. Let $s_1(t)$ be the set of addresses of the decayed cells in PUF_{id_1} and similarly $s_2(t)$ for PUF_{id_2}. The similarity between PUF_{id_1} and PUF_{id_2} is expressed as

$$J^{1,2}_{\text{inter}}(t) = J(s_1(t), s_2(t)). \tag{2}$$

A small value of $J^{1,2}_{\text{inter}}$ indicates high uniqueness. Our DRAM PUFs exhibit almost perfect behavior, with J_{inter} values for decay times up to 60 s that do not exceed 0.001 for the Intel Galileo and 0.003 for the PandaBoard. At higher decay times, up to 6 minutes, Intel Galileo exhibits a maximum of $J_{\text{inter}} = 0.007$ at $t = 360$ s. The PandaBoard shows larger values with a maximum of 0.041 at $t = 300$ s, which is still close to the optimal value of zero. Those comparably low values at shorter decay times are due to the fact that

4. A key feature of Linux, the so-called `workqueues`, allowing tasks to be scheduled at specific time intervals, is used for this purpose.

5. One required change is disabling or limiting the journaling service. Other options available are to reduce the size of the journal, or using persistent storage for the journal.

TABLE 2
Metrics for Logical PUF Instances Measured at Different Decay Times $\mathcal{T}_1$ and $\mathcal{T}_2$ as Well as for Different PUF Sizes

$\mathcal{T}_1 = \{10\,\text{s}, 20\,\text{s}, 30\,\text{s}, 40\,\text{s}, 50\,\text{s}, 60\,\text{s}\}$, size $= 16$ MB						$\mathcal{T}_2 = \{120\,\text{s}, 180\,\text{s}, 240\,\text{s}, 300\,\text{s}, 360\,\text{s}\}$, size $= 32$ KB					
decay times $\mathcal{T}_1$	device type	min. J_{intra}	max. J_{inter}	H_t (bits)	avg. number decayed cells	decay times $\mathcal{T}_2$	device type	min. J_{intra}	max. J_{inter}	H_t (bits)	avg. number decayed cells
10 s	PandaBoard	6.870×10^{-1}	0.000	7.062×10^{3}	5.250×10^{2}	120 s	PandaBoard	4.630×10^{-1}	1.000×10^{-2}	1.362×10^{4}	1.069×10^{3}
	IntelGalileo	3.850×10^{-1}	0.000	3.810×10^{2}	2.300×10^{1}		IntelGalileo	7.710×10^{-1}	4.000×10^{-3}	3.382×10^{3}	2.450×10^{2}
20 s	PandaBoard	7.120×10^{-1}	3.000×10^{-4}	6.741×10^{4}	6.132×10^{3}	180 s	PandaBoard	4.380×10^{-1}	1.700×10^{-2}	3.163×10^{4}	2.675×10^{3}
	IntelGalileo	4.750×10^{-1}	0.000	3.837×10^{3}	2.720×10^{2}		IntelGalileo	8.360×10^{-1}	4.000×10^{-3}	8.482×10^{3}	6.400×10^{2}
30 s	PandaBoard	7.260×10^{-1}	1.000×10^{-3}	2.294×10^{5}	2.380×10^{4}	240 s	PandaBoard	4.090×10^{-1}	2.600×10^{-2}	4.736×10^{4}	4.161×10^{3}
	IntelGalileo	4.650×10^{-1}	1.000×10^{-3}	1.508×10^{4}	1.194×10^{3}		IntelGalileo	6.260×10^{-1}	5.000×10^{-3}	1.381×10^{4}	1.085×10^{3}
40 s	PandaBoard	7.380×10^{-1}	1.000×10^{-3}	5.099×10^{5}	5.835×10^{4}	300 s	PandaBoard	4.220×10^{-1}	4.100×10^{-2}	5.911×10^{4}	5.307×10^{3}
	IntelGalileo	5.140×10^{-1}	4.000×10^{-4}	4.266×10^{4}	3.711×10^{3}		IntelGalileo	7.940×10^{-1}	6.000×10^{-3}	1.960×10^{4}	1.588×10^{3}
50 s	PandaBoard	7.620×10^{-1}	2.000×10^{-3}	8.973×10^{5}	1.108×10^{5}	360 s	PandaBoard	3.480×10^{-1}	3.400×10^{-2}	6.738×10^{4}	6.129×10^{3}
	IntelGalileo	5.500×10^{-1}	2.000×10^{-4}	8.658×10^{4}	8.078×10^{3}		IntelGalileo	8.280×10^{-1}	7.000×10^{-3}	2.912×10^{4}	2.444×10^{3}
60 s	PandaBoard	7.690×10^{-1}	3.000×10^{-3}	1.374×10^{6}	1.805×10^{5}						
	IntelGalileo	5.880×10^{-1}	4.000×10^{-4}	1.478×10^{5}	1.459×10^{4}						

Left: results for decay times $\mathcal{T}_1 = \{10\,\text{s}, 20\,\text{s}, 30\,\text{s}, 40\,\text{s}, 50\,\text{s}, 60\,\text{s}\}$ *and* size $= 16$ *MB. Right: results for decay times* $\mathcal{T}_2 = \{120\,\text{s}, 180\,\text{s}, 240\,\text{s}, 300\,\text{s}, 360\,\text{s}\}$ *and* size $= 32$ *KB.*

many fewer DRAM cells have had the chance to decay within these short time periods (see Fig. 4). As given in Table 2, the values for both configurations suggest that both device types exhibit high uniqueness, with the Intel Galileo showing inherently smaller J_{inter} values compared to the values from the PandaBoard.

Robustness. Consider again the experiment where a DRAM-PUF is given time t to decay. Let $s(t)$ be the set of addresses of decayed cells in one run of this experiment, and $s'(t)$ in a subsequent run of the experiment on the same PUF_{id}. We characterize the robustness of the PUF as

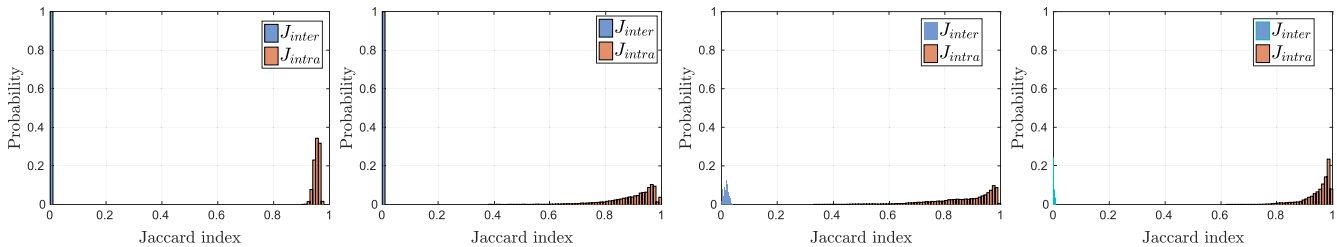
$$J_{\text{intra}}(t) = J(s(t), s'(t)). \tag{3}$$

Large values of J_{intra} indicate high robustness. Fig. 3 shows the distributions of J_{intra} and J_{inter} for different decay times. A wide gap between the two distributions indicates that individual devices can be distinguished perfectly. Note that at shorter decay times we observe minimum J_{intra} values of 0.385 for the Intel Galileo, whereas for the PandaBoard noise values are smaller with the minimal J_{intra} value 0.687 at $t = 10$ s (see Table 2). The differences in the J_{intra} values among the two groups of configurations reflect variations of ambient conditions (i.e., temperature) over time. Nevertheless, we note that in both cases, the J_{intra} values are high enough to allow the devices to be used successfully. We can therefore conclude that our devices constitute robust PUF behavior in a realistic usage scenario, where ambient conditions, such as temperature, are expected to naturally differ.

Entropy. If we want to generate cryptographic keys from the PUF responses, the PUFs must exhibit sufficient entropy. We estimate the entropy of DRAM PUFs in the following manner. We again consider the observed set $s(t)$ of indices of DRAM cells that have decayed by time t. The cardinality of $s(t)$ is denoted as $l_t = |s(t)|$, and N is the total number of DRAM cells. We assume that each DRAM cell independently has a probability $p(t)$ of having a decay time smaller than t (such that it usually decays in time less than t). We estimate $p(t)$ as l_t/N. The PUF entropy associated with time t is given by

$$H_t = Nh(p(t)) \approx Nh(l_t/N), \tag{4}$$

where $h(p) = p \log \frac{1}{p} + (1-p)\log \frac{1}{1-p}$ is the binary entropy function. A single observation of $s(t)$ may not be sufficient for determining $p(t)$ because of short-term noise. Thus, we are estimating $p(t)$ by averaging 50 observations of l_t, i.e., we are computing multiple measurements. Table 2 lists the entropy H_t as bits per measured logical PUF (i.e., 16 MB and 32 KB). We observe that the entropy is significantly higher on the PandaBoard, correlating with the higher number of bit flips of this device type. This is most likely due to the different technologies used to implement DRAM cells. In particular, the results show that the minimum entropy of



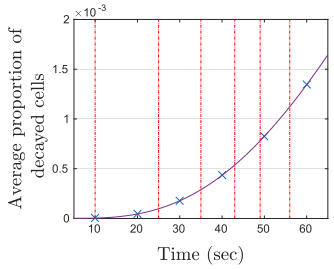
PandaBoard results, obtained at all the decay times in the set $\mathcal{T}_1$, using size $= 16$ MB.

Intel Galileo results, obtained at all the decay times in the set $\mathcal{T}_1$, using size $= 16$ MB.

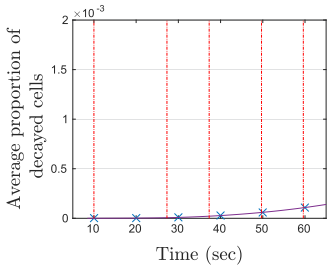
PandaBoard results, obtained at all the decay times in the set $\mathcal{T}_2$, using size $= 32$ KB.

Intel Galileo results, obtained at all the decay times in the set $\mathcal{T}_2$, using size $= 32$ KB.

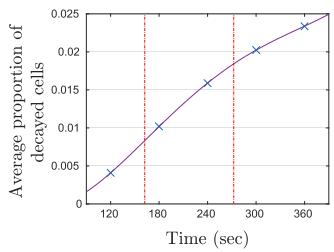
Fig. 3. Histograms of J_{intra} and J_{inter} values for multiple instances of the PandaBoard and the Intel Galileo, (left two graphs) for decay times $\mathcal{T}_1 = \{10\,\text{s}, 20\,\text{s}, 30\,\text{s}, 40\,\text{s}, 50\,\text{s}, 60\,\text{s}\}$ and size $= 16$ MB and (right two graphs) for decay times $\mathcal{T}_2 = \{120\,\text{s}, 180\,\text{s}, 240\,\text{s}, 300\,\text{s}, 360\,\text{s}\}$ and size $= 32$ MB.



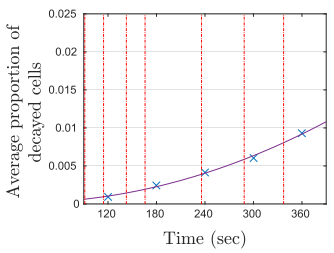
PandaBoard, measured at decay times in the set $\mathcal{T}_1$ and $\texttt{size} = 16$ MB.



Intel Galileo, measured at decay times in the set $\mathcal{T}_1$ and $\texttt{size} = 16$ MB.



PandaBoard, measured at decay times in the set $\mathcal{T}_2$ and $\texttt{size} = 32$ KB.



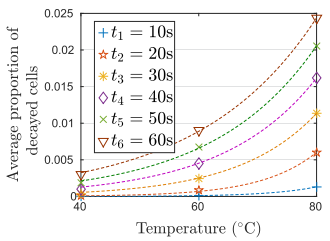
Intel Galileo, measured at decay times in the set $\mathcal{T}_2$ and $\texttt{size} = 32$ KB.

Fig. 4. Proportion of decayed DRAM cells as a function of time for multiple instances of the PandaBoard and the Intel Galileo, (left two graphs) for decay times $\mathcal{T}_1 = \{10\text{ s}, 20\text{ s}, 30\text{ s}, 40\text{ s}, 50\text{ s}, 60\text{ s}\}$ and $\texttt{size} = 16$ MB and (right two graphs) for decay times $\mathcal{T}_2 = \{120\text{ s}, 180\text{ s}, 240\text{ s}, 300\text{ s}, 360\text{ s}\}$ and $\texttt{size} = 32$ KB. Possible challenge times are indicated by vertical lines.

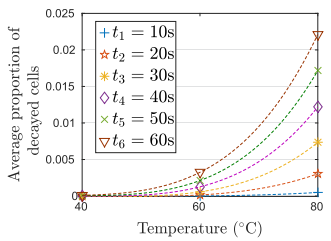
the PandaBoard can be up to one order of magnitude larger compared to the Intel Galileo, i.e., at $t = 240$ s the PandaBoard provides 25949 bits per 32 KB of DRAM, versus 9692 bits for the Intel Galileo. At larger t more DRAM cells get a chance to decay, increasing l_t and hence the entropy. However, large values of t make PUF handling too slow to be practical.

Regarding the fractional entropy, the values of the proposed DRAM PUF are orders of magnitude smaller, compared to SRAM PUFs, which usually have 0.7 to 0.9 bits of entropy per cell. However, DRAM is usually orders of magnitude larger than SRAM, and can provide enough entropy in total.

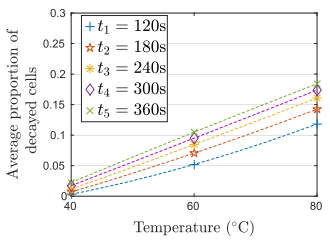
Decay Dependency on Time and Temperature. Fig. 4 shows the average proportion of decayed cells, l_t/N, as a function of time t. All measurements were taken at (ambient) room temperature with DRAM chips operating at around 40 °C. Every point in the plot represents an average taken over all logical PUFs. We see that the number of decayed cells significantly increases with time.



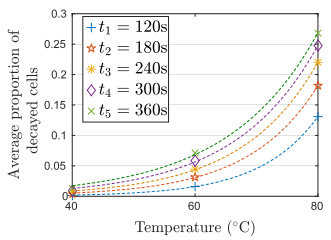
PandaBoard results, obtained at decay times in the set $\mathcal{T}_1$, using $\texttt{size} = 16$ MB.



Intel Galileo results, obtained at decay times in the set $\mathcal{T}_1$, using $\texttt{size} = 16$ MB.



PandaBoard results, obtained at decay times in the set $\mathcal{T}_2$, using $\texttt{size} = 32$ KB.



Intel Galileo results, obtained at decay times in the set $\mathcal{T}_2$, using $\texttt{size} = 32$ KB.

Fig. 5. Proportion of decayed DRAM cells as a function of temperature for multiple instances of the PandaBoard and the Intel Galileo, (top) for decay times $\mathcal{T}_1 = \{10\text{ s}, 20\text{ s}, 30\text{ s}, 40\text{ s}, 50\text{ s}, 60\text{ s}\}$ and $\texttt{size} = 16$ MB and (bottom) for decay times $\mathcal{T}_2 = \{120\text{ s}, 180\text{ s}, 240\text{ s}, 300\text{ s}, 360\text{ s}\}$ and $\texttt{size} = 32$ KB.

This plot allows us to estimate the number of time-dependent challenges that a logical PUF can support. In order to allow for unique identification at different decay times, the set of decay times $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$ must be chosen such that the corresponding measurements taken at decay time t_{x+1} show a minimum number of *new* bit flips $\epsilon_{t_x} = l_{t_{x+1}} - l_{t_x}$, with respect to the previous one t_x, which must be greater than the inherent noise. Given the noise values and ϵ_t, the set of viable decay times and thus the challenges of a logical PUF can be determined accordingly. We computed a conservative, minimum number of possible challenges per logical PUF, by using the maximum noise (i.e., minimum J_{intra} value) and the minimum number of bit flips, previously observed at each decay time t. We experimentally determined the maximum number of challenges for decay times $\mathcal{T}_1$ and PUF $\texttt{size} = 16$ MB to be $n = 5$ for the Intel Galileo and $n = 6$ for the PandaBoard, as well as $n = 7$ and $n = 2$, respectively for $\mathcal{T}_2$ and PUF $\texttt{size} = 32$ KB. Possible challenge times are indicated by vertical red lines in Fig. 4.

A second factor influencing the number of decayed DRAM cells is temperature. Fig. 5 shows the proportion of decayed cells as a function of temperature. Temperatures lower than room temperature were achieved using a thermal chamber. All other temperatures were stablized using a ceramic heater circuit. We observed that heating (or cooling) the DRAM affects all cells in the same way: the decay is accelerated (or slowed down) by the same factor. For example, at temperature $T' > T$ it is possible to find a decay time $t' < t$ such that $s(t')_{T'} = s(t)_T$, i.e., from a heated DRAM, operating at temperature T', a similar response can be obtained in time t' as from a cooler DRAM in time t. The adapted decay time t' at an increased temperature T' that results in a similar decay behavior as using decay time t at temperature T (with $T' > T$), can be computed as

$$t'_{T'} = t \cdot e^{-\alpha(T'-T)}. \tag{5}$$

Based on our measurements, we estimated α to be 0.068 for the Intel Galileo platform and 0.066 for the PandaBoard.

Fig. 6 shows the Jaccard index J_{intra} between measurements $s(t)_T$ at (ambient) room temperature with DRAM operating at $T = 40\,^{\circ}$C for $t = \{120\text{ s}, 180\text{ s}, 240\text{ s}, 300\text{ s}, 360\text{ s}\}$ and measurements $s(t')_{T'}$ performed on the same DRAM PUF, at temperatures $T' = \{10^{\circ}\text{C}, 20^{\circ}\text{C}, 30^{\circ}\text{C}, 40^{\circ}\text{C}, 50^{\circ}\text{C}, 60^{\circ}\text{C}, 70^{\circ}\text{C}, 80^{\circ}\text{C}\}$ for adjusted $t' = \{t_1, t_2, t_3, t_4, t_5\}$. For $T' = 40\,^{\circ}$C, $t' = t$, whereas a distinct set t' of adjusted time points exists for each

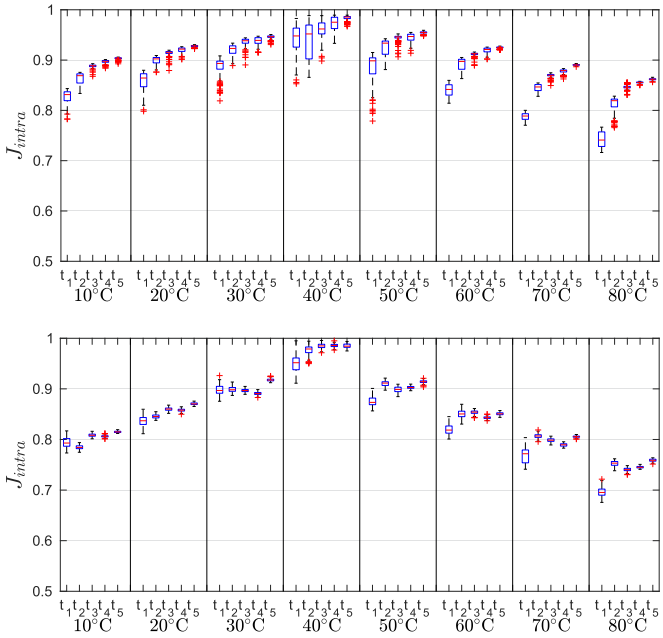


Fig. 6. J_{intra} values (i.e., similarity) of enrollment measurements taken at $T = 40\,^{\circ}\text{C}$ and reconstruction measurements at temperatures $T' = \{10\,^{\circ}\text{C}, 20\,^{\circ}\text{C}, 30\,^{\circ}\text{C}, 40\,^{\circ}\text{C}, 50\,^{\circ}\text{C}, 60\,^{\circ}\text{C}, 70\,^{\circ}\text{C}, 80\,^{\circ}\text{C}\}$, with adjusted decay times t' for multiple instances of the PandaBoard (top) and the Intel Galileo (bottom).

different temperature of the set T', such that $s(t')_{T'} = s(t)_{40\,^{\circ}\text{C}}$. J_{intra} remains higher than 0.65, indicating reconstruction at different temperature is feasible. Especially, when the reconstruction temperature is close to the enrollment temperature, the noise is small. This confirms that differences in temperature can effectively be accommodated by adjusting t and that the PUF behavior at different temperatures can be predicted.

Stability Over Time. During extended lifetime of devices, DRAM aging effects will begin to take place. Existing work on SRAM PUFs has explored aging effects [24], [25], [48], [49]. We are only aware of limited work on aging-related effects in DRAM cells with regard to security [50]. Fig. 7 shows the histogram of J_{intra} values for measurements of both evaluation boards, taken roughly 16 months apart. Note that the measurements also include the noise introduced by temperature changes in our lab. The values for the Intel Galileo and the PandaBoard are similar to the J_{intra} results shown in Table 2, suggesting sufficient stability of DRAM PUFs over a long-term usage time period.

5 A Helper Data System for DRAM PUFs

5.1 Helper Data System Construction

In order to use a PUF for key storage a Helper Data System, also known as Fuzzy Extractor [51], [52], [53], [54], is required. The HDS takes care of the noise in the PUF measurements and ensures that a cryptographic key can reproducibly be generated from the PUF. An HDS consists of an enrollment algorithm `Enroll` and a reconstruction algorithm `Rec`. The algorithm `Enroll` probes the PUF and outputs a secret key $\mathcal{K}$ and helper data w. The helper data is stored in nonvolatile memory. It must be assumed that an adversary has access to w as it is stored publicly on the device. Therefore, the `Enroll` algorithm is crafted such that w does not leak information about $\mathcal{K}$. At reconstruction

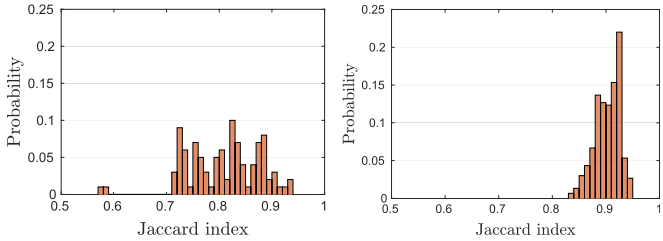


Fig. 7. Distribution of J_{intra} values computed between measurements pairs, taken at enrollment and reconstruction from the same logical PUF instances, over ≈ 16 months apart. Values are shown for the PandaBoard left and Intel Galileo right.

time, algorithm `Rec` does a fresh measurement of the PUF, reads w and computes a best guess $\hat{\mathcal{K}}$ for the key from those two inputs. If the HDS is properly designed and the noise was below a certain threshold, then $\hat{\mathcal{K}} = \mathcal{K}$.

We construct an HDS that is similar to HDSs for SRAM PUFs [55]. The main difference is that the decay behavior of DRAM cells is an analog property, while SRAM startup states are discrete. We discretise the decay properties by giving DRAM cells a label '`F`' (fast) or '`S`' (slow) or no label. The `F` cells lose their charge quickly, the `S` cells slowly, and form sets $\mathcal{F}$ and $\mathcal{S}$ respectively. Our `Enroll` algorithm creates two layers of helper data: (i) pointers to memory cells which (during enrollment) were either extremely fast or extremely slow; (ii) helper data for the Code Offset Method (COM) [52], [53], [56], [57], [58]. The first layer improves noise resilience by selecting only those cells that are unlikely to change their decay speed significantly when environmental conditions change. Using pointers which identify a similar number of stable `F` and `S` cells also avoids the problem of *high bias*, as there are many more `S` cells than `F` cells in a DRAM module. In particular, the Code Offset Method, one of the simplest and most popular Helper Data Systems, becomes ineffective when its input has a large bias towards either 0 or 1, as shown in [59], [60]. In the case of large bias, PUF measurements exhibit decreased entropy, and in turn the COM's helper data w leaks information about the key $\mathcal{K}$. In contrast, our use of pointers was introduced in [60] and is similar to 'Index Based Syndromes' [61], which avoids leakage problems due to bias. The details of the HDS follow.

5.2 System Setup

We present our enrollment algorithm in Algorithm 1 and our reconstruction algorithm in Algorithm 2. A linear error correcting code is chosen that encodes k-bit messages in n-bit codewords. The syndrome function of the code is $\texttt{Syn} : \{0,1\}^n \rightarrow \{0,1\}^{n-k}$. Syndrome decoding is denoted as $\texttt{SynDec} : \{0,1\}^{n-k} \rightarrow \{0,1\}^n$. A key derivation function $\texttt{KeyDeriv} : \{0,1\}^n \times \{0,1\}^* \rightarrow \{0,1\}^\ell$ is chosen, which takes as input an n-bit secret string and public randomness, and outputs an ℓ-bit secret key. System parameters n_f and n_s, with $n_s + n_f = n$, and parameters N, t_1, t_2, t_3 are fixed. The parameters t_1, t_2 and t_3 are decay times, chosen such that $t_1 \leq t_3$. In particular, values N, t_1, t_3 are chosen such that the enrollment can be performed in a limited amount of time $Nt_1 + t_3$ while still accurately labeling cells as `F` and `S` cells. Time t_2 is chosen such that (a) reconstruction is sufficiently fast and (b) noise is small: a fast cell has had more time to discharge than at the enrollment timescale t_1, and a slow cell has had less time to discharge than at enrollment

timescale t_3. In Section 5.3 we provide values for the parameters, based on the decay characteristics of our evaluation platforms. The parameter ℓ is set such that any information about x that the adversary may have (due to bias or non-uniform distributions of F and S cells, correlations between memory cells, etc.) is squeezed out by KeyDeriv's compression and does not end up in $\mathcal{K}$. The KeyDeriv algorithm can, for example, be a universal hash function [62], [63] or a q-wise independent hash function [64].

Algorithm 1. Enroll

1 For $i \in \{1, \ldots, N\}$ do the following experiment:
  Charge the DRAM. Let it decay for time t_1. Let $\mathcal{F}_i$ be the set of addresses of the decayed cells;
2 $\mathcal{F} = \mathcal{F}_1 \cap \cdots \cap \mathcal{F}_N$;
3 Charge the DRAM. Let it decay for time t_3;
4 Let $\mathcal{S}$ be the set of addresses of the cells that have not yet decayed;
5 Randomly pick n_f elements from $\mathcal{F}$ and n_s elements from $\mathcal{S}$, with $n_f + n_s = n$. Construct a vector r by putting the n elements in a random order.
  Construct $x \in \{0,1\}^n$ such that $x_i = 1$ if $r_i \in \mathcal{F}$ and $x_i = 0$ if $r_i \in \mathcal{S}$;
6 $w = \mathtt{Syn}(x)$;
7 Generate random p. Compute $\mathcal{K} = \mathtt{KeyDeriv}(x, p)$;
8 Store (r, w, p) in memory.

In the reconstruction procedure, the device may perform a temperature measurement and then adjust t_2 to the temperature using a formula similar to Equation (5).

There are well known methods by which Rec can verify if the reconstructed key is correct and if the stored data has been manipulated [54]. They are omitted here for the sake of brevity. In step 1 of Algorithm 2 we write (r', w', p') because the stored data may have been manipulated.

Step 4 of Algorithm 2 uses the linearity of the error correcting code. The expression $w' \oplus \mathtt{Syn}(x')$ equals $\mathtt{Syn}(x \oplus x')$, i.e., the syndrome of the error vector. The SynDec outputs the error vector, which is then xor'ed into x' to reconstruct x.

Algorithm 2. Rec

1 Read (r', w', p');
2 Charge the DRAM. Let it decay for time t_2;
3 Construct $x' \in \{0,1\}^n$ such that $x'_i = 1$ if the cell at address r'_i is decayed and 0 otherwise;
4 $\hat{x} = x' \oplus \mathtt{SynDec}(w' \oplus \mathtt{Syn}(x'))$;
5 $\hat{\mathcal{K}} = \mathtt{KeyDeriv}(\hat{x}, p')$.

5.3 Experimental Validation of the HDS

To find realistic reference values for the parameters of the HDS, we validated the proposed Helper Data System for its practicability, based on the device types which we used in Section 4. The parameter t_2 should be chosen such that only with small probability F cells will not decay at t_2 and similarly, that the decay of S cells at t_2 is unlikely. In order to estimate the noise, we evaluated fractional bit error rates (BER): BER_1 for F cells observed at t_1 and BER_2 for S cells observed at t_3, respectively. The maximum of BER_1 and BER_2 indicates the noise in the reconstruction. While both bit error rates can take values ranging from zero to one,

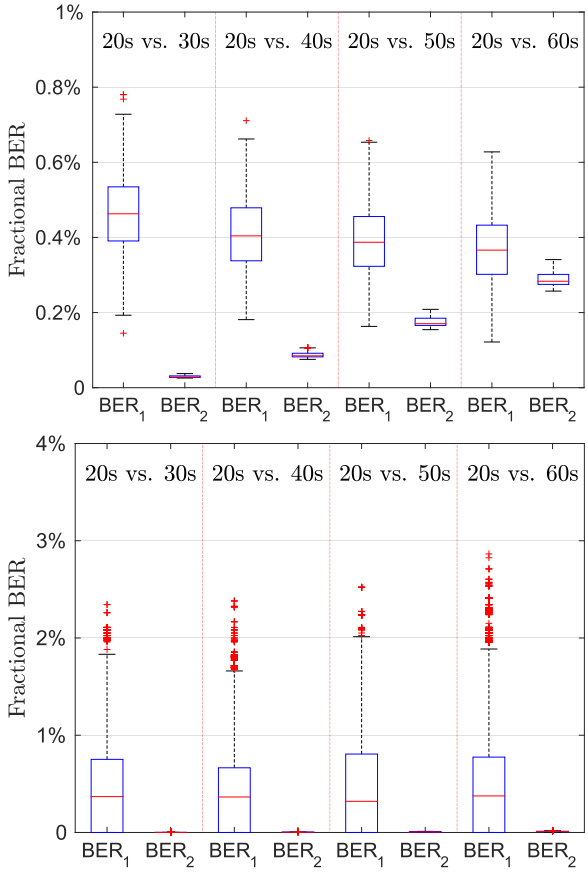


Fig. 8. Fractional bit error rates BER_1 and BER_2 (in %) computed between subsequent decay times for PandaBoard (top) and Galileo (bottom).

preferable BERs are close to zero. BER_1 indicates the proportion of F cells, which only decayed at t_1 but not at t_2, normalized by the total amount of cells, which decayed until t_1

$$\mathrm{BER}_1 = \frac{|s(t_1) \setminus s(t_2)|}{l_{t_1}}. \tag{6}$$

In contrast, BER_2 gives the proportion of S cells, which decayed at t_2 but did not decay at enrollment decay time t_3, normalized by the number of all the cells that comprise a measurement, except those ones flipped at t_3. BER_2 is computed as

$$\mathrm{BER}_2 = \frac{|s(t_2) \setminus s(t_3)|}{\mathcal{N} - l_{t_3}}. \tag{7}$$

In the rest of this section, we evaluate a more efficient enrollment procedure, which requires only one enrollment measurement, by setting $t_1 = t_3 = 20$ s and $N = 1$ to estimate bit error rates. In particular, we compared enrollment measurement m_e taken at $t_1 = t_3 = 20$ s with various reconstruction measurements m_r obtained at decay times $t_2 = \{30\,\mathrm{s}, 40\,\mathrm{s}, 50\,\mathrm{s}, 60\,\mathrm{s}\}$. We obtained values for BER_1 and BER_2 between (m_e, m_r) pairs. Both bit error rates, computed for different t_2 decay times and for both evaluation platforms, are given in Fig. 8. The very small number of fractional bit errors, with a maximum of 0.8 percent for the PandaBoard and 2.9 percent for the Intel Galileo, support the computation of helper data as described in Section 5.1.

The observed numbers of bit flips for a 16 MB memory region at different decay times, in combination with the low bit error rates presented above, allow for using only a single enrollment measurement, i.e., setting $t_1 = t_3$. In particular, we observed several orders of magnitude more S cells than F cells, up to decay time t_3. Moreover, the small error rates shown in Fig. 8 suggest that only a small percentage of cells in $\mathcal{N} \setminus \mathcal{F}$ decay faster than expected (with $\mathcal{N}$ being the set of all DRAM cells), which allows us to label the fast cells with a single enrollment and safely assume that all the other cells are slow. For example, if a 128 bit key was to be reconstructed, based on the evaluation results, we can set $t_1 = t_3 = 20$ s on the Intel Galileo, resulting in an average of 272 F cells, with a worst-case bit error of 3 percent. On the PandaBoard, we can set $t_1 = t_3 = 10$ s, which leads to an average of 525 F cells and a maximum bit error rate of 2 percent.[6]

In order to robustly derive a cryptographic key on the basis of the error rates given in Fig. 8, we employ a linear error-correcting code, such as a (n, k, t) BCH code. For example, considering a maximum error rate of 3 percent and a message length of $m = 128$ bit (i.e., a typical AES key), a $(31, 26, 1)$ BCH code, which operates on 5 concatenated blocks, requires 155 flipped DRAM bits as input. For all the 16 MB PUF regions we measured, enough bit flips appear in the enrollment with decay time $t_1 = 20$ s on Intel Galileo and $t_1 = 10$ s on PandaBoard. After the enrollment measurement, the PUF size can be adjusted to have just enough bit flips, thus allowing for multiple PUF instances within one DRAM. As an example, given the average number of decayed cells of the Intel Galileo at decay time $t_1 = 20$ s and PUF size $= 16$ MB, listed in Table 2, one requires only ≈ 9 MB to store a 128 bit key. In contrast, the PandaBoard requires ≈ 4.7 MB when using an enrollment decay time of $t_1 = 10$ s. As the number of bit flips varies across different DRAMs in a product line, to enroll DRAMs in a product line, the number of bit flips versus decay time should be measured on a few samples, and the enrollment decay time should then be chosen accordingly, such as 20 s for Galileo. Then the PUF size can be further chosen after the enrollments are done, to ensure there are enought bit flips.

The described HDS introduces a novel solution regarding error correction for the DRAM decay PUF examined in [33]. It is considerably compact, as the helper data employed requires only minimal memory space. Additionally, it is very simple to implement and can work for a biased PUF, as proven by our experimental validation.

6 A Lightweight Authentication Protocol

If a device supports the computation of helper data, as described in Section 5, it can immediately provide stable PUF keys for use in any symmetric or asymmetric cryptographic protocol. In this section we consider the case of a highly resource-constrained device which does have DRAM, but not the processing power to run the key reconstruction phase of the Helper Data System. Especially the SynDec function can be computation-intensive.

6. Note that these values are only valid for the average case, using the evaluated device types. Using other device types may lead to different minimum number of bit flips that must be taken care of when setting values for the parameters of the HDS.

In this lightweight scenario, the PUF device is also unable to perform any cryptographic operation. If we want to construct an authentication scheme based on PUF responses, then the parties will inevitably have to transmit information about PUF responses *in plaintext*. This makes all lightweight protocols susceptible to Man-In-The-Middle (MITM) attacks. Nevertheless it makes sense to implement lightweight authentication, as it presents a cost-effective hurdle against 'casual' attacks.

We present a lightweight PUF-based mutual authentication protocol for this scenario in Algorithm 3. The protocol is based on the mutual comparison of sets s_x which contain indices of decayed cells at increasing time scales t_x. One party reveals the set s_x, randomly contaminated with indices pointing to undecayed cells. The other party demonstrates its ability to identify which indices belong to s_x.

Requirements. A prover device $\mathcal{P}$ needs access to some resource offered by a verifier $\mathcal{V}$, and has to prove that it possesses a specific logical PUF (PUF_{id}). Furthermore, $\mathcal{P}$ trusts the resource only if $\mathcal{V}$ too knows the responses of PUF_{id}.

Consider an active attacker whose aim is to obtain PUF responses by pretending to be one of the parties. We want to build our scheme in the following way. If the attacker impersonates the PUF device $\mathcal{P}$, the protocol should force him to be the first party to provide information about the PUF response. Thus the attacker does not easily get access to the resources that $\mathcal{V}$ is protecting; i.e., the attacker first needs to learn PUF responses. If on the other hand the attacker impersonates $\mathcal{V}$, it should not be easy for him to *quickly* extract all the PUF responses from $\mathcal{P}$. In order to satisfy this requirement, we make sure that the initiative to start the protocol lies with $\mathcal{P}$. In this way the attacker has to wait until $\mathcal{P}$ initiates contact.

Attacker Model. We consider an adversary who is able to observe the communication between $\mathcal{P}$ and $\mathcal{V}$, and also to engage in a protocol exchange with either $\mathcal{P}$ or $\mathcal{V}$. We do not consider man-in-the-middle attacks or message modification. The protocol is publicly known, including all the system parameters.

System Setup. A vector $\mathcal{T} = \{t_0, t_1, \ldots, t_n\}$ of decay times (with $t_0 < t_1 < \cdots < t_n$) is carefully chosen such that $\forall_x \, l_{t_{x+1}} - l_{t_x} = \epsilon_{t_x}$, i.e., at every time step the number of newly decayed cells always equals the security parameter ϵ_t.

The set of enrollment times $\mathcal{T}_{\mathrm{enroll}} = \{t_0^e, t_1^e, \ldots, t_n^e\}$ is chosen to evaluate the BER and set system parameters Δ_1 and Δ_2. Here, BER_1 and BER_2 are calculated according to Equations (6) and (7) by setting $t_2 = t_i^e$ and $t_1 = t_3 = t_i$. Furthermore, thresholds are set as $\Delta_1 = \mathrm{BER}_1$, $\Delta_2 = \max(\mathrm{BER}_1, \mathrm{BER}_2)$. For example, for $t_i = 20$ s, if $t_i^e = 30$ s, $\mathrm{BER}_1 = 1\%$ for PandaBoard and 3 percent for Intel Galileo; if $t_i^e = 20$ s, $\mathrm{BER}_1 = 6\%$ for PandaBoard and 53 percent for Intel Galileo.

Enrollment. Enrollment is conducted by a trusted party $\mathcal{SYS}$, such as a manufacturer or a system integrator. $\mathcal{SYS}$ queries the PUF at decay times $\mathcal{T}_{\mathrm{enroll}}$ and gets a set of measurements for each PUF_{id}: $\mathcal{M}_{id} = \{s_e^{id}(t_0), s_e^{id}(t_1), \ldots, s_e^{id}(t_n)\}$. The sets $\mathcal{M}_{id}$ are distributed over multiple verifiers, considering that different verifiers must not share the same id. For each PUF_{id} the prover device initializes the counter c_{id} to zero, and the verifier initializes the counter c'_{id} to zero.

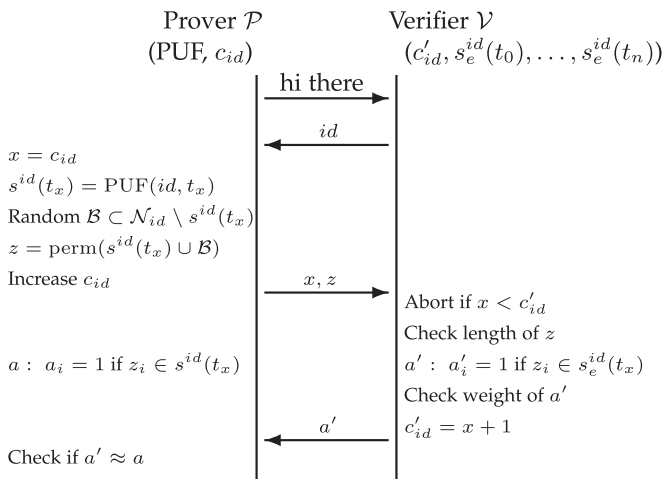


Fig. 9. The lightweight mutual authentication protocol.

Algorithm 3. `Mutual Authentication` Let $\mathcal{N}_{id}$ Denote the Set of all Memory Cells in PUF_{id}.

1 $\mathcal{P}$ initiates contact;
2 $\mathcal{V}$ sends id to $\mathcal{P}$;
3 $\mathcal{P}$ performs the following actions:
Set $x = c_{id}$. Perform a measurement of PUF_{id} at decay time t_x. The result is a set of addresses $s^{id}(t_x)$ of decayed cells. Randomly select addresses into a set $\mathcal{B} \subset \mathcal{N}_{id} \setminus s^{id}(t_x)$ of size $|\mathcal{B}| = 2\epsilon_t(x+1) - l_x^{id}$ and construct a vector z by randomly permuting $s^{id}(t_x) \cup \mathcal{B}$. Construct a bit string $a \in \{0,1\}^{2\epsilon_t(x+1)}$ such that $a_i = 1$ if $z_i \in s^{id}(t_x)$ and $a_i = 0$ otherwise. Increase c_{id} and send x, z to $\mathcal{V}$;
4 $\mathcal{V}$ performs the following actions:
Continue only if $x \geq c'_{id}$ and z has length $2\epsilon_t(x+1)$; else abort. Construct $a' \in \{0,1\}^{2\epsilon_t(x+1)}$ such that $a'_i = 1$ if $z_i \in s_e^{id}(t_x)$. If the fractional Hamming weight of a' is larger than $\frac{1}{2}(1-\Delta_1)$, then set $c'_{id} = x+1$ and send a', else abort;
5 $\mathcal{P}$ checks if the fractional Hamming distance between a and a' is smaller than Δ_2. If not, $\mathcal{P}$ aborts.

Mutual Authentication. After $\mathcal{P}$ and $\mathcal{V}$ establish contact in the first two steps of Algorithm 3, the prover constructs the address vector z from the addresses ($s^{id}(t_x)$) of decayed cells and a random set ($\mathcal{B}$) of addresses that have not yet decayed. Given that $\mathcal{P}$ has knowledge about the temperature behavior of his DRAM-PUF, he can use a temperature-scaled decay time t' (see Equation (5)) in order to retrieve z. The random permutation ensures that attackers cannot derive $s^{id}(t_x)$ from z. The selection of the random set $\mathcal{B}$ ensures that the protocol is not hindered by the potentially large number of erroneous bit flips, even if the probability of such an error is small per cell (see Section 5.3), the huge number of cells in a logical PUF may drive up the number of bit errors. Note that $\mathcal{P}$ adjusts the size of $\mathcal{B}$ so that z has size $2\epsilon_t(x+1)$.

Due to the tuning of the string length, the string a is balanced, i.e., it contains approximately as many '0's as '1's, ensuring large entropy of a given z. Letting multiple instances of the protocol run, we assume the attacker to know the locations of flipped bits at previous decay times. In particular, an eavesdropper Eve knows $l_{x-1}^{id} \approx \epsilon_t \cdot x$ addresses that also appear in $s^{id}(t_x)$. Hence, the number of addresses unknown to Eve is $\approx \epsilon_t$. The entropy of a given z is then the entropy of ϵ_t positions out of $(x+1)2\epsilon_t - x\epsilon_t$, i.e., $\log \binom{(x+2)\epsilon_t}{\epsilon_t}$, which can be approximated as $\epsilon_t(x+2)h(\frac{1}{x+2}) \geq \epsilon_t$.

Note that $\mathcal{P}$ keeps track of c_{id}, otherwise an attacker could impersonate a verifier and learn the complete memory state for each id, t_x by communicating with $\mathcal{P}$ many times. Furthermore, $\mathcal{V}$ also has to keep track of c'_{id}, otherwise an attacker could replay a z from the past. The check if $x \leq c'_{id}$ is meant to detect replays. In step 4 of Algorithm 3, the verifier performs a check on the Hamming weight of a'. This verifies if $\mathcal{P}$ is authentic. If z is sent by an impostor then with very high probability z will not contain $\epsilon_t(1-\Delta_1)$ addresses that are also in the enrollment $s_e^{id}(t_x)$. In step 5, $\mathcal{P}$ checks the Hamming distance between a and a', concluding the mutual authentication protocol.

The prover device uses every pair (id, x) only once. As soon as $\mathcal{P}$ sends a string z, it increases its counter c_{id}. This is independent of the a versus a' verification at the verifier side. Note that an attacker can pretend to be $\mathcal{P}$ and make many attempts to authenticate to $\mathcal{V}$ without affecting c'_{id}.

Moreover, there is a straightforward denial of service attack. The attacker can repeatedly pretend to be a verifier and abort at step 4 of the protocol. With each aborted run, $\mathcal{P}$ is forced to increase the counter x. At some point PUF_{id} is exhausted. However, as $\mathcal{P}$ is the party that initiates the protocol, the attacker cannot set the pace of his denial of service attack. Furthermore, $\mathcal{P}$ can be programmed to (temporarily) stop communicating if it observes consecutive failures. A sequence diagram of the protocol is depicted in Fig. 9.

7 Conclusion

In this work we presented intrinsic PUFs that can be extracted from Dynamic Random-Access Memory in commodity devices. An evaluation of the DRAM PUFs found on unmodified, commodity devices, in particular the PandaBoard and Intel Galileo, showed their robustness, uniqueness, randomness, and stability over period of several months. Moreover, in contrast to existing DRAM and SRAM PUFs, decay-based DRAM PUFs can be queried directly during run-time. We further presented an HDS scheme tailored towards DRAM PUFs as well as a lightweight protocol for device authentication that draws its security from time-dependent decay characteristics of our DRAM PUF. Our intrinsic DRAM PUFs overcome two limitations of the popular intrinsic SRAM PUFs: they have the ability to be accessed at run-time, and have an expanded challenge-response space due to the use of a decay time t that is part of the challenge. Consequently, our work presents a new alternative for device authentication by leveraging DRAM in commodity devices.

Acknowledgments

This work has been partly funded by DFG as part of project P3 within the CRC 1119 CROSSING, and also by the German Academic Exchange Service (Deutscher Akademischer Austauschdienst - DAAD). This work was also supported in part by the US National Science Foundation (NSF) under NSF Grant no. 1651945. A. Schaller and W. Xiong contributed equally to this work.

REFERENCES

[1] G. Hernandez, O. Arias, D. Buentello, and Y. Jin, "Smart nest thermostat: A smart spy in your home," *Black Hat USA*, 2014. [Online]. Available: http://blackhat.com/docs/us-14/materials/us-14-Jin-Smart-Nest-Thermostat-A-Smart-Spy-In-Your-Home-WP.pdf, Accessed on: Mar. 10, 2018.

[2] P. Venda, Pen Test Partners LLP, "Hacking DefCon 23's IoT Village Samsung fridge." Aug. 18, 2015. [Online]. Available: https://www.pentestpartners.com/blog/hacking-defcon-23s-iot-village-samsung-fridge, Accessed on: Jul. 8, 2016.

[3] A. Greenberg, "Hackers remotely kill a jeep on the highway—with me in it," *Wired (online)*. Jul. 21, 2015 [Online]. Available: https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/, Accessed on: Jul. 8, 2016.

[4] I. Foster, A. Prudhomme, K. Koscher, and S. Savage, "Fast and vulnerable: A story of telematic failures," in *Proc. USENIX Workshop Offensive Technol.*, 2015, pp. 15–15.

[5] B. Schneier, "The internet of things is wildly insecure—and often unpatchable," *Wired (online)*. Jan. 6, 2014 [Online]. Available: http://www.wired.com/2014/01/theres-no-good-way-to-patch-the-internet-o f-things-and-thats-a-huge-problem/, Accessed on: Jul. 8, 2016.

[6] J. Viega and H. Thompson, "The state of embedded-device security (Spoiler alert: It's bad)," *IEEE Secur. Privacy*, vol. 10, no. 5, pp. 68–70, Sep./Oct. 2012.

[7] F. Armknecht, R. Maes, A.-R. Sadeghi, B. Sunar, and P. Tuyls, "Memory leakage-resilient encryption based on physically unclonable functions," in *Towards Hardware-Intrinsic Security*. Berlin, Germany: Springer, 2010, pp. 135–164.

[8] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *Proc. Des. Autom. Conf.*, 2007, pp. 9–14.

[9] Ü. Kocabaş, A. Peter, S. Katzenbeisser, and A.-R. Sadeghi, "Converse PUF-Based Authentication," in *Int. Conf. Trust and Trustworthy Computing*, Berlin, Germany: Springer, pp. 142–158, 2012.

[10] P. Tuyls and L. Batina, "RFID-tags for anti-counterfeiting," in *Topics in Cryptology*. Berlin, Germany: Springer, 2006, pp. 115–131.

[11] J. Guajardo, S. S. Kumar, G.-J. Schrijen, and P. Tuyls, "FPGA intrinsic PUFs and their use for IP protection," in *Cryptographic Hardware and Embedded Systems*, Berlin, Germany: Springer, pp. 63–80, 2007.

[12] J. Guajardo, S. S. Kumar, G. J. Schrijen, and P. Tuyls, "Brand and IP protection with physical unclonable functions," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2008, pp. 3186–3189.

[13] A. Schaller, T. Arul, V. van der Leest, and S. Katzenbeisser, "Lightweight anti-counterfeiting solution for low-end commodity hardware using inherent PUFs," in *Int. Conf. Trust and Trustworthy Computing*, Berlin, Germany: Springer, 2014, pp. 83–100.

[14] F. Kohnhäuser, A. Schaller, and S. Katzenbeisser, "PUF-based software protection for low-end embedded devices," in *Int. Conf. Trust and Trustworthy Computing*, Berlin, Germany: Springer, 2015, pp. 3–21.

[15] R. A. Scheel and A. Tyagi, "Characterizing composite user-device touchscreen physical unclonable functions (PUFs) for mobile device authentication," in *Proc. Int. Workshop Trustworthy Embedded Devices*, 2015, pp. 3–13.

[16] J. Kong, F. Koushanfar, P. K. Pendyala, A.-R. Sadeghi, and C. Wachsmann, "PUFatt: Embedded platform attestation based on novel processor-based PUFs," in *Proc. ACM/EDAC/IEEE Des. Autom. Conf.*, 2014, pp. 1–6.

[17] S. Schulz, A.-R. Sadeghi, and C. Wachsmann, "Short paper: Lightweight remote attestation using physical functions," in *Proc. ACM Conf. Wireless Netw. Secur.*, 2011, pp. 109–114.

[18] P. Tuyls and B. Škorić, "Secret key generation from classical physics: Physical uncloneable functions," in *AmIware Hardware Technology Drivers of Ambient Intelligence*. Berlin, Germany: Springer, 2006, pp. 421–447.

[19] B. Škorić, G.-J. Schrijen, P. Tuyls, T. Ignatenko, and F. Willems, *Secure key storage with PUFs*, pp. 269–292, 2007.

[20] B. Gassend, D. Clarke, M. Van Dijk, and S. Devadas, "Delay-based circuit authentication and applications," in *Proc. ACM Symp. Appl. Comput.*, 2003, pp. 294–301.

[21] Intrinsic ID B.V., "Intrinsic-ID to showcase TrustedSensor IoT security solution at InvenSense Developers Conference," [Online]. Available: https://www.intrinsic-id.com/intrinsic-id-to-showcase-trustedsensor-iot-security-solution-at-invensense-developers-conference/, Accessed on: Jul. 8, 2016.

[22] V. van der Leest, "SBIR project: Bring your own security," NCSRA Symposium. 2015 [Online]. Available: https://www.dcypher.nl/files/Intrinsic-ID.pdf, Accessed on: Jul. 8, 2016.

[23] G.-J. Schrijen and V. van der Leest, "Comparative analysis of SRAM memories used as PUF primitives," in *Proc. Conf. Des., Autom. Test Eur.*, 2012, pp. 1319–1324.

[24] G. Selimis, M. Konijnenburg, M. Ashouei, J. Huisken, H. De Groot, V. Van der Leest, G.-J. Schrijen, M. Van Hulst, and P. Tuyls, "Evaluation of 90 nm 6T-SRAM as physical unclonable function for secure key generation in wireless sensor nodes," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2011, pp. 567–570.

[25] R. Maes, V. Rožić, I. Verbauwhede, P. Koeberl, E. Van der Sluis, and V. Van der Leest, "Experimental evaluation of physically unclonable functions in 65 nm CMOS," in *Proc. European Solid-State Circuits Conf. (ESSCIRC)*, 2012, pp. 486–489.

[26] S. Katzenbeisser, Ü. Kocabaş, V. Rožić, A.-R. Sadeghi, I. Verbauwhede, and C. Wachsmann, "PUFs: Myth, fact or busted? A security evaluation of physically unclonable functions (PUFs) cast in silicon," in *Proc. Int. Workshop Cryptographic Hardware Embedded Syst.*, 2012, pp. 283–301.

[27] A. Bacha and R. Teodorescu, "Authenticache: Harnessing cache ECC for system authentication," in *Proc. Int. Symp. Microarchitecture*, 2015, pp. 128–140.

[28] S. Rosenblatt, S. Chellappa, A. Cestero, N. Robson, T. Kirihata, and S. S. Iyer, "A self-authenticating chip architecture using an intrinsic fingerprint of embedded DRAM," *IEEE J. Solid-State Circuits*, vol. 48, no. 11, pp. 2934–2943, Nov. 2013.

[29] F. Tehranipoor, N. Karimian, K. Xiao, and J. Chandy, "DRAM based intrinsic physical unclonable functions for system level security," in *Proc. Great Lakes Symp. VLSI*, 2015, pp. 15–20.

[30] F. Tehranipoor, N. Karimian, W. Yan, and J. A. Chandy, "DRAM-based intrinsic physically unclonable functions for system-level security and authentication," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 25, no. 3, pp. 1085–1097, Mar. 2017.

[31] S. Sutar, A. Raha, and V. Raghunathan, "D-PUF: An intrinsically reconfigurable DRAM PUF for device authentication in embedded systems," in *Proc. Int. Conf. Compilers, Architectures, Sythesis Embedded Syst. (CASES)*, 2016, pp. 1–10.

[32] C. Keller, F. Gurkaynak, H. Kaeslin, and N. Felber, "Dynamic memory-based physically unclonable function for the generation of unique identifiers and true random numbers," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2014, pp. 2740–2743.

[33] 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 *Proc. Conf. Cryptographic Hardware Embedded Syst.*, Aug. 2016, pp. 432–453.

[34] P. Prabhu, A. Akel, L. M. Grupp, S. Y. Wing-Kei, G. E. Suh, E. Kan, and S. Swanson, "Extracting device fingerprints from flash memory by exploiting physical variations," in *Proc. Int. Conf. Trust Trustworthy Comput.*, 2011, pp. 188–201.

[35] Y. Wang, W.-K. Yu, S. Wu, G. Malysa, G. E. Suh, and E. C. Kan, "Flash memory for ubiquitous hardware security functions: True random number generation and device fingerprints," in *Proc. IEEE Symp. Secur. Privacy*, 2012, pp. 33–47.

[36] S. Rosenblatt, D. Fainstein, A. Cestero, J. Safran, N. Robson, T. Kirihata, and S. S. Iyer, "Field tolerant dynamic intrinsic chip ID using 32 nm high-K/metal gate SOI embedded DRAM," *IEEE J. Solid-State Circuits*, vol. 48, no. 4, pp. 940–947, Apr. 2013.

[37] W. Liu, Z. Zhang, M. Li, and Z. Liu, "A trustworthy key generation prototype based on DDR3 PUF for wireless sensor networks," *Sensors*, pp. 11 542–11 556, 2014.

[38] M. S. Hashemian, B. Singh, F. Wolff, D. Weyer, S. Clay, and C. Papachristou, "A robust authentication methodology using physically unclonable functions in DRAM arrays," in *Proc. Des. Autom. Test Eur. Conf.*, 2015, pp. 647–652.

[39] A. Rahmati, M. Hicks, D. E. Holcomb, and K. Fu, "Probable cause: The deanonymizing effects of approximate DRAM," in *Proc. Int. Symp. Comput. Archit.*, 2015, pp. 604–615.

[40] B. Keeth, R. J. Baker, B. Johnson, and F. Lin, *DRAM Circuit Design: Fundamental and High-Speed Topics*. Hoboken, NJ, USA: John Wiley & Sons, 2008.

[41] U. Rührmair, J. Sölter, and F. Sehnke, "On the foundations of physical unclonable functions," *IACR Cryptology ePrint Archive*, 2009. [Online]. Available: https://eprint.iacr.org/2009/277, Accessed on: Jul. 8, 2016.

[42] C. A. Waldspurger, "Memory resource management in VMware ESX server," *ACM SIGOPS Operating Syst. Rev.*, vol. 36, pp. 181–194, 2002.
[43] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors," *ACM SIGARCH Comput. Archit. News*, vol. 42, pp. 361–372, 2014.
[44] 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 *Proc. IEEE Int. Symp. Hardware Oriented Secur. Trust*, 2017, pp. 1–7.
[45] F. Tehranipoor, N. Karimian, W. Yan, and J. A. Chandy, "Investigation of DRAM PUFs reliability under device accelerated aging effects," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2017, pp. 1–4.
[46] J. Liu, B. Jaiyen, Y. Kim, C. Wilkerson, and O. Mutlu, "An experimental study of data retention behavior in modern DRAM devices: Implications for retention time profiling mechanisms," *ACM SIGARCH Comput. Archit. News*, vol. 41, pp. 60–71, 2013.
[47] P. Jaccard, "Etude comparative de la distribution florale dans une portion des Alpes et du Jura," *Bulletin de la Societe Vaudoise des Sciences Naturelles*, vol. 37, pp. 547–579, 1901.
[48] R. Maes and V. van der Leest, "Countering the effects of silicon aging on SRAM PUFs," in *Proc. IEEE Int. Symp. Hardware-Oriented Secur. Trust*, 2014, pp. 148–153.
[49] A. Schaller, B. Škorić, and S. Katzenbeisser, "On the systematic drift of physically unclonable functions due to aging," in *Proc. Int. Workshop Trustworthy Embedded Devices*, 2015, pp. 15–20.
[50] B. Schroeder, E. Pinheiro, and W.-D. Weber, "DRAM errors in the wild: A large-scale field study," in *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 37, pp. 193–204, 2009.
[51] J.-P. Linnartz and P. Tuyls, "New shielding functions to enhance privacy and prevent misuse of biometric templates," in *Int. Conf. Audio-and Video-Based Biomet. Person Authentication*, Berlin, Germany: Springer, 2003, pp. 393–402.
[52] Y. Dodis, L. Reyzin, and A. Smith, "Fuzzy extractors: How to generate strong keys from biometrics and other noisy data," in *Advances in Cryptology—EUROCRYPT*, 2004, pp. 523–540.
[53] Y. Dodis, R. Ostrovsky, L. Reyzin, and A. Smith, "Fuzzy Extractors: How to generate strong keys from biometrics and other noisy data," *SIAM J. Comput.*, vol. 38, no. 1, pp. 97–139, 2008.
[54] X. Boyen, "Reusable cryptographic fuzzy extractors," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2004, pp. 82–91.
[55] C. Bösch, J. Guajardo, A.-R. Sadeghi, J. Shokrollahi, and P. Tuyls, "Efficient helper data key extractor on FPGAs," in *Proc. Int. Workshop Cryptographic Hardware Embedded Syst.*, 2008, pp. 181–197.
[56] C. Bennett, G. Brassard, C. Crépeau, and M. Skubiszewska, "Practical quantum oblivious transfer," in *Advances in Cryptology—CRYPTO*, 1991, pp. 351–366.
[57] A. Juels and M. Wattenberg, "A fuzzy commitment scheme," in *Proc. ACM Conf. Comput. Commun. Secur.*, 1999, pp. 28–36.
[58] B. Škorić and N. de Vreede, "The spammed code offset method," *IEEE Trans. Inf. Forensics Secur.*, vol. 9, no. 5, pp. 875–884, May 2014.
[59] R. Maes, V. van der Leest, E. van der Sluis, and F. Willems, "Secure key generation from biased PUFs," in *Proc. Int. Workshop Cryptographic Hardware Embedded Syst.*, 2015, pp. 517–534.
[60] B. Škorić, "A trivial debiasing scheme for helper data systems," *IACR Cryptology ePrint Archive*, 2016. [Online]. Available: https://eprint.iacr.org/2016/241, Accessed on: Jul. 8, 2016.
[61] M.-D. Yu and S. Devadas, "Secure and robust error correction for physical unclonable functions," *IEEE Des. Test Comput.*, vol. 27, no. 1, pp. 48–65, Jan./Feb. 2010.
[62] J. L. Carter and M. N. Wegman, "Universal classes of hash functions," *J. Comput. Syst. Sci.*, vol. 18, no. 2, pp. 143–154, 1979.
[63] D. Stinson, "Universal hashing and authentication codes," *Des. Codes Cryptography*, vol. 4, pp. 369–380, 1994.
[64] Y. Dodis, K. Pietrzak, and D. Wichs, "Key derivation without entropy waste," in *Advances in Cryptology—EUROCRYPT*, 2014, pp. 93–110.

André Schaller received the PhD degree for his research regarding lightweight applications for intrinsic Physically Unclonable Functions (PUFs) on commodity devices, from TU Darmstadt, Germany, in 2017. His research interests include hardware-based and embedded security, with a special focus on PUFs.

Wenjie Xiong received the BSc degree in microelectronics and psychology from Peking University, China, in 2014. She is currently working toward the PhD degree in the Department of Electrical Engineering, Yale University, USA, under Prof. Jakub Szefer. Her research interests include physically unclonable functions, physical cryptography, and security verification of processor architectures.

Nikolaos Athanasios Anagnostopoulos (S'18) received the BSc degree in computer science from the Aristotles University of Thessaloniki, Greece, in 2012, the MSc degree in computer science from the University of Twente, the Netherlands, and the MSc degree in innovation in information and communication technology from TU Berlin, Germany, in 2014, and is currently working toward the PhD degree in the Security Engineering Group, Technical University of Darmstadt, Germany. His research interests include hardware security, with a focus on embedded devices, PUFs and IoT. He is a student member of the IEEE.

Muhammad Umair Saleem received the BSc degree in electronics engineering from the Bahauddin Zakariya University Multan, Pakistan, in 2012, and the MSc degree in Information and Communication Engineering from TU Darmstadt, Germany, in 2018. His research interests include embedded systems, internet of things, automation, and embedded security.

Sebastian Gabmeyer received the PhD degree on model checking based verification techniques for graph transformations from the Vienna University of Technology, Austria, in 2015, and was with the Security Engineering Group in TU Darmstadt, Germany, until late 2017. His research interests include hardware security and software verification.

Boris Škorić received the Ph.D. degree in theoretical physics from the University of Amsterdam, The Netherlands, in 1999. From 1999 to 2008, he was a research scientist with Philips Research, The Netherlands, working first on display physics and later on security topics. In 2008, he joined the Department of Mathematics and Computer Science, Eindhoven University of Technology, The Netherlands, as an Assistant Professor.

Stefan Katzenbeisser (S'98-A'01–M'07-SM'12) received the PhD degree from the Vienna University of Technology, Austria. After working as a research scientist with the Technical University of Munich, Germany, he joined Philips Research as a senior scientist in 2006. Since 2008, he has been a professor with the Technical University of Darmstadt, heading the Security Engineering Group. His current research interests include embedded security, data privacy, and cryptographic protocol design. He has authored more than 200 scientific publications and served on the program committees of several workshops and conferences devoted to information security. He is currently serving on the Information Forensics and Security Technical Committee of the IEEE Signal Processing Society. He is a senior member of the IEEE.

Jakub Szefer received the BSc (highest honors) degree in electrical and computer engineering from the University of Illinois at Urbana-Champaign, and the MA and PhD degrees in electrical engineering from Princeton University where he worked with Prof. Ruby B. Lee on secure hardware architectures. He joined Yale University in summer 2013 as an assistant professor of electrical engineering, where he started the Computer Architecture and Security Laboratory (CAS Lab). His research interests include the intersection of computer architecture, system software, and hardware security. His research focuses on secure hardware-software architectures for servers and mobile devices, virtualization and cloud security, hardware security verification, physically unclonable functions, and hardware FPGA implementation of cryptographic algorithms. His research is supported through National Science Foundation and industry donations.