

Run-time Accessible DRAM PUFs in Commodity Devices

Wenjie Xiong¹, André Schaller², Nikolaos A. Anagnostopoulos²,
Muhammad Umair Saleem², Sebastian Gabmeyer²,
Stefan Katzenbeisser², and Jakub Szefer¹

¹ Yale University, New Haven, CT, USA
{wenjie.xiong,jakub.szefer}@yale.edu

² Technische Universität Darmstadt and CASED, Darmstadt, Germany
{schaller,anagnostopoulos}@seceng.informatik.tu-darmstadt.de
{gabmeyer,katzenbeisser}@seceng.informatik.tu-darmstadt.de
muhammadumair.saleem@stud.tu-darmstadt.de

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 fabrication processes. Prior works have demonstrated that PUFs are a promising cryptographic primitive to enable secure key storage, hardware-based device authentication and identification. So far, most PUF constructions require addition of new hardware or FPGA implementations 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 is the first to enable the run-time access of decay-based intrinsic DRAM PUFs in commercial off-the-shelf systems, which requires no additional hardware or FPGAs. A key advantage of our PUF construction is that it can be queried during run-time of a Linux system. Furthermore, by exploiting different decay times of individual DRAM cells, the challenge-response space is increased. Finally, we introduce lightweight protocols for device authentication and secure channel establishment, that leverage the DRAM PUFs at run-time.

1 Introduction

Continued miniaturization and cost reduction of processors and System-on-Chip designs have enabled the creation of almost ubiquitous smart devices, from smart thermostats and refrigerators, to smartphones and embedded car entertainment systems. While there are numerous advantages to the proliferation of such smart devices, they create new security vulnerabilities [12, 1, 8, 6]. One major concern is that they often lack the implementation of sufficient security mechanisms [34, 46]. Critical challenges in securing these devices are to provide 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.

A classic 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 [2]. In order to address these issues, researchers have proposed Physically Unclonable Functions (PUFs). PUFs leverage the unique behavior of a device due to manufacturing variations as a hardware-based fingerprint. A PUF instance is extremely difficult to replicate, even by the manufacturer. Hence, PUFs have been proposed as cryptographic building blocks in security primitives and protocols for: authentication and identification [40, 18, 43], hardware-software binding [9, 10, 31, 19, 33], remote attestation [20, 37], and secret key storage [45, 44]. So far, most types of PUFs in digital electronic systems (such as arbiter PUFs [7, 40]) require 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 [9], which are PUFs that are already inherent to a device.

Intrinsic PUFs are considered an attractive low-cost security anchor, as they provide PUF instances within standard hardware that can be found in commercial off-the-shelf devices [42, 26], without requiring any hardware modifications. The most prominent example of an intrinsic PUF is a PUF based on Static Random-Access Memory (SRAM) [31, 19, 35, 38, 25], which draws its characteristics from the startup values of bi-stable SRAM memory cells. SRAM PUFs are known to have good PUF characteristics [14]. However, PUF measurements must be extracted during a very early boot stage (before the SRAM is used). Consequently, the derived key can only be used at this time, or must be saved to a different memory region, which may cause security problems. Recently, a new error-based SRAM PUF, which can be accessed at run-time, was proposed [3]. However, to query the PUF, the supply voltage needs to be lowered to induce errors in SRAM cells, requiring special hardware in the processor.

Most recently, PUF-like behavior has been found in Dynamic Random-Access Memory (DRAM) [28]. One approach to extract unique DRAM behavior induced by manufacturing variations relies on startup tendencies of DRAM cells [41]. Another approach to extract DRAM PUFs is to leverage the unique decay characteristics of DRAM cells. In [16], authors exploit the fact that charges of individual DRAM cells, if not refreshed, decay over time in a unique manner. PUF responses³ can be generated by initializing the 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 unique bit flips at unique locations, which in their entirety can be used as a PUF response by reading the DRAM content after the decay time elapsed. However, current state of the art requires custom hardware or FPGA-based platforms [41, 16], in order to modify the DRAM refresh mechanism such that DRAM PUF extraction is possible.

This paper is the first to extract DRAM PUFs from commercial systems, requiring no special hardware modifications or FPGA setup, and to provide a

³ In the rest of the paper we will use the terms PUF response and PUF measurement interchangeably.

practical solution to query DRAM PUFs during run-time on a Linux system. Our decay-based DRAM PUF allows for repeated access, which overcomes the limitation of previous intrinsic memory-based PUFs that were available at device startup only. Moreover, the capacity of DRAM is magnitudes larger than SRAM, allowing to draw many more bits in order to derive larger cryptographic key material, or to segment DRAM into several logical PUFs. Furthermore, DRAM is an excellent candidate for an intrinsic PUF as DRAM is an integral part of today’s commodity platforms and can be found in many “smart” devices, such as smartphones or smart thermostats. Recent use of embedded DRAM [29, 4] in low-cost microprocessors will further increase the availability of DRAM as part of mobile and embedded computing platforms.

1.1 Related Work on DRAM PUFs

The earliest approach to exploit manufacturing variations of DRAM cells for identification and random number generation was reported in [28, 29], in which an embedded DRAM chip was designed to generate fingerprints to mitigate hardware counterfeiting. In subsequent work [16, 22, 27], the decay of external DDR3 modules was evaluated through memory controllers in FPGAs and was used for identification and key storage. Other work has focused on the design of a circuit exploiting the variation in write reliability of DRAM cells [11], and presented an authentication scheme based on signatures generated using such variations. Unlike our work, all previous research required dedicated circuits to be designed or FPGAs to be used. To the best of our knowledge, this is the first work to present an approach to enable the usage of intrinsic DRAM PUF instances on commodity devices at run-time. Further, we provide a system-level solution for querying the DRAM PUF while a Linux OS is running on same hardware and is actively using DRAM chip wherein the PUF is located.

1.2 Contributions

- We extract decay-based DRAM PUF instances from unmodified commodity devices, including the PandaBoard and the Intel Galileo platforms. Two approaches are presented: (i) accessing the PUF at device startup using a customized firmware, and (ii) querying the PUF with a kernel module, while Linux OS is running on same hardware and is actively using DRAM chip wherein the PUF is located.
- Through extensive experiments, we show that DRAM PUFs exhibit robustness, reliability, and in particular allow usage of the decay time as part of the PUF challenge.
- We introduce new metrics for evaluating DRAM PUFs, based on the Jaccard index, and show they are significantly better suited for the decay-based DRAM PUF evaluation over the classic Hamming-distance based metrics.
- Finally, we exploit time-dependent decay characteristics of DRAM cells in the design of PUF-enhanced protocols. In particular, we show protocols for device identification and authentication that draw their security from the time-dependent decay of DRAM cells.

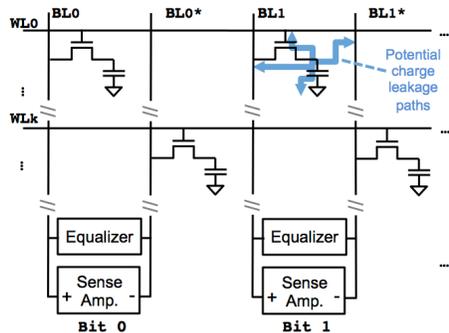


Fig. 1: A single DRAM cell consists of a capacitor and a transistor, connected to word-lines (WL) and bit-lines (BL and BL*); arrows indicate leakage paths for dissipation of charges that lead to PUF behavior.

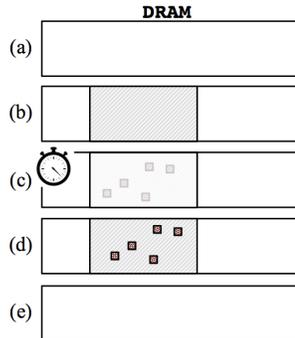


Fig. 2: Five steps required for runtime access of a DRAM PUF. Only during steps (b) – (d) the memory associated with the PUF is not usable for other processes.

1.3 Paper Organization

The remainder of the paper is organized as follows. Section 2 presents background on DRAM and introduces our DRAM PUFs. Section 3 describes our experimental setup and the implementation of software needed to realize the DRAM PUFs. Section 4 contains our evaluation of DRAM PUFs characteristics. Section 5 describes lightweight protocols for device authentication and secure channel establishment. Section 6 presents open research issues. Section 7 concludes the paper.

2 Extracting DRAM PUFs from Commodity Devices

In a DRAM cell, a single data bit is stored in a capacitor and can be accessed through a transistor, as shown in Figure 1. DRAM cells are grouped in arrays, where each row of the array is connected to a horizontal word-line. Cells in the same column are connected to a bit-line. All bit-lines are coupled to sense-amplifiers that amplify small voltages on bit-lines to levels such that they can be interpreted as logical zeros or ones. In order to access a row, all the bit-lines will be *precharged* to half the supply voltage $V_{DD}/2$; subsequently the word-line is *enabled*, connecting every capacitor in that row with its bit-line. The sense amplifier will then drive the bit-line to V_{DD} or $0V$, depending on the charge on the capacitor. The amplifiers are usually shared by two bit-lines [15], 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 $0V$ in the capacitor, whilst 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 its 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 32ms or 64ms. Without this periodic refresh, some of the cells will slowly decay to 0, while others decay to 1, depending on whether they are a true cell or an anti-cell. Because of the manufacturing variations among DRAM cells, some cells decay faster than others, which can be exploited as PUF.

2.1 Decay-based PUFs in DRAM

The process of exploiting the unique decay behavior of DRAM cells to extract a PUF measurement is summarized in Figure 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 memory region defined by starting address (*addr*) and size (*size*) is reserved, e.g., using memory ballooning introduced in Section 2.2. Furthermore, the refresh for the PUF region is disabled and the initialization value (*initval*) 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. (d) After the decay time has expired, 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 operating system (OS) again.

We introduce here the concept of logical DRAM PUFs, which are memory regions within a DRAM module that are used for obtaining the PUF measurements. For a particular DRAM, each logical PUF is determined by: (i) *addr*, the starting address of the logical PUF, and (ii) *size*, its size. A typical DRAM memory can be divided into thousands or more logical PUFs.

To get a challenge/response, two additional parameters are needed. First, an initial value (*initval*), which is written into the cells in the DRAM region before any decay. Second, the desired decay time (*t*) that will cause enough charge to have leaked in some cells such that their stored logical bits will flip. As the decay time and the positions of the flipped bits are unique for individual DRAM regions, the “pattern” of flipped bits for a given decay time *t* can serve 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, *initval*, 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 initial 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 the true cells and anti-cells, and hence the individual decay directions), it is possible to construct an initialization pattern 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 have to be stored on the device. In our evaluation, we use a fixed initialization value *initval* = 0 to all cells. The entropy of our measurements thus can be further improved.

Overall, the challenge of a DRAM PUF can be defined as a tuple (id, t) , where id denotes the logical PUF instance ($addr$ and $size$) and t denotes the decay time after which the memory content is read. We will not specify the *interval* as we assume it is fixed.

Although SRAM and DRAM PUFs are both considered weak PUFs [30], 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} , taken at corresponding 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. With increasing decay times t , the number of DRAM cells flipping is monotonically increasing. Thus, m_{x+1} consists of a number of *newly* flipped bits as well as the majority⁴ 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 $m_x \subseteq m_{x+1}$, up to noise. However, note that it is not possible to measure responses for several decay times t_0, t_1, \dots, 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 the data is read from DRAM cells into row buffers). Querying a PUF response with different decay time thus requires one to restart the experiment.

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 with Yocto Linux crashes about a minute after DRAM refresh is disabled. Therefore, we present a customized solution which allows us to refresh critical code, but leaves PUF areas untouched. This solution is based on two techniques dubbed *selective DRAM refresh* and *memory ballooning*. The former allows for selectively refreshing the 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 OS and user-space programs accesses, to let the cells decay during PUF measurement.

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 querying 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 in the granularity of the entire DRAM⁵. Refresh at segment granularity

⁴ 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.

⁵ 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

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 0V, depending on the value that was in the cell. In this way, the capacitor charge is restored to the value it had before the charges leaked. 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. 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 (*initval*) 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 [47]. 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 memory region that will be reserved, i.e., the PUF. Once PUF memory is “ballooned”, DRAM refresh can be disabled and selective refresh enabled for the non-PUF memory region. After PUF access is finished, 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 has less capabilities in doing so due to the fact that disabling and enabling DRAM refresh includes writing to hardware registers, a task which can only be performed by the kernel. An attacker thus requires root privileges. Furthermore, accessing the memory dedicated to the PUF is restricted to the kernel as well. Thus, a crucial security assumption is that firmware and operating system are trusted and an attacker never gains root privileges.

An attacker may try to change the ambient temperature in order to influence the bit flip characteristics, but 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 [17], i.e., inducing random bit flips into DRAM cells by repeatedly accessing adjacent rows. However, he or she would not succeed, as DRAM PUFs allocate a continuous chunk of memory. Rowhammering would only apply at the borders of the PUF area. Using voltage variations in order to manipulate PUF behavior, as done in [11, 28], is out of scope of this paper, as we are focussing on intrinsic PUFs on commodity hardware where such voltage control is not possible.

to store the most significant 8 bits of every 16 bits, while the other chip is used to store the least significant 8 bits of every 16 bits. Disabling refresh on a single chip is not possible, as half of each memory word would be lost.

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 1.2V. The Intel Galileo has an Intel Quark SoC X1000 SoC with two 128MB 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 is easy to implement 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 the Galileo. 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. This makes it ideal for gathering PUF measurements.

In the case of the Galileo platform, we modified the Quark EDKII firmware. PUF measurement code was inserted just before DRAM refresh is enabled in order to access the PUF, comprising the following steps: writing the initial value (*initval*) 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. Most of the code implements initializing 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: DRAM is initialized with *initval*, 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 cells in the desired logical PUF region by the *initval*. (2) The kernel module then modifies the memory controller via writes to configuration registers to disable DRAM refresh, while memory holding the OS and application is selectively refreshed. (3) After the decay time of t seconds 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 512MB. Thus, 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 case of the PandaBoard, in order to implement this configuration, the interleaving mechanism that alternately maps subsequent logical addresses to physical addresses from both modules must first be disabled within the boot-loader. 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 are residing functional on the other DRAM module. The kernel module takes about 100 lines of C code in total.

On the Intel Galileo, the refresh of the whole DRAM has to be disabled as it is not possible to control refresh at granularity smaller than a DRAM module. Consequently, the kernel module must selectively refresh memory used by the kernel and applications. The kernel module schedules selective refresh tasks⁶ 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 core 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 much longer than the required refresh period, critical portions of code and data may have decayed before they can be accessed by the kernel module.

Table 1 shows the time required to perform selective refresh of memory regions of various sizes, ranging from 32MB up to 128MB. We see that selective refresh takes between 7.6ms and 21.2ms for a single run. The last two columns in Table 1 show the CPU time spent on selective refresh, assuming 64ms and 200ms refresh rates. For an active memory size of 128MB, the system will spend 33% of CPU time on selective refresh, when a target refresh period of 64ms is selected. However, at room temperature, the 64ms refresh period, picked by most vendors, is very conservative, and our experiments suggest that even with a refresh rate of 200ms our setup is stable. Previous work on DRAM retention time support our results [21]. Thus, depending on the operating conditions and required stability guarantees, the selective refresh period can be increased, allowing larger DRAM

⁶ A key feature of Linux, the so-called `workqueues`, allowing tasks to be scheduled at specific time intervals, is used for this purpose.

Table 1: Time needed to perform memory reads (i.e. the selective refresh) to refresh varying sizes of memory regions on the Intel Galileo board with DDR3 memory.

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

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 32MB without any special modifications.⁷ At 32MB, only 7.6ms are needed for selective refresh at 64ms, making more than 87% CPU time available for other applications. These numbers demonstrate that selective refresh is viable for realistic code sizes.

4 Evaluation of DRAM PUF Characteristics

We measured the PUF instances on the Intel Galileo and PandaBoard, as described in Section 3. We performed measurements using four different PandaBoards and five Intel Galileo devices. Furthermore, given the large amount of memory present, we measured two 32KB 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 five different decay times t , with 50 measurements each. Based on these measurements we evaluated robustness, uniqueness, and randomness, as well as time and temperature dependency, and stability of the DRAM PUFs.

The characteristics of the DRAM PUFs are different compared to the SRAM PUFs. Rather than being considered as an array of bits, a DRAM PUF response mainly comprises the positions of flipped bits in a memory region. Thus, classic metrics that are used to evaluate memory-based PUFs, which are usually based on fractional Hamming distances, do not properly reflect the properties of the DRAM PUFs. This effect is particularly noticeable when evaluating the uniqueness of PUF instances. In case of the SRAM PUF, uniqueness is expressed due to differences in the startup values of all SRAM cells amongst different devices. Consequently, uniqueness is measured using the fractional Hamming distance between startup values taken from different PUF instances (inter-Hamming Distance, inter-HD). However, in case of the DRAM PUF it is rather the location, i.e., the indices, of the cells that flip, which is the root cause for the uniqueness. If one would apply the *fractional* inter-Hamming Distance, the whole 32KB

⁷ One required change is disabling or limiting the journaling service. Other options available are to reduce the size of the journal so it does not take much memory, or using persistent storage for the journal.

Table 2: Metrics of logical PUF instances measured at different decay times.

decay time	device family	min.	max.	fractional	avg.	max.	min.
		J_{intra}	J_{inter}	entropy H_t/N	decay rate	fractional intra-HD	fractional inter-HD
120s	PandaBoard	0.4634	0.0102	0.0271	0.0041	0.0045	0.0038
	IntelGalileo	0.7712	0.0038	0.0062	0.0009	0.0003	0.0012
180s	PandaBoard	0.4382	0.0168	0.0754	0.0102	0.0083	0.0139
	IntelGalileo	0.8361	0.0044	0.0169	0.0024	0.0005	0.0032
240s	PandaBoard	0.4087	0.0258	0.0893	0.0159	0.0101	0.0244
	IntelGalileo	0.6261	0.0049	0.0250	0.0041	0.0020	0.0057
300s	PandaBoard	0.4222	0.0405	0.1478	0.0202	0.0123	0.0238
	IntelGalileo	0.7944	0.0055	0.0353	0.0061	0.0013	0.0080
360s	PandaBoard	0.3484	0.0342	0.1440	0.0234	0.0206	0.0279
	IntelGalileo	0.8276	0.0072	0.0541	0.0093	0.0022	0.0124

measurement would be considered, including those cells that did not flip within the observed time period, resulting in a very low value, which does not capture uniqueness to the full extent.

Thus, we propose new metrics to evaluate robustness and uniqueness of the DRAM PUF that are based on the Jaccard index[13]. The Jaccard index is a well known metric to quantify the similarity of two sets of different size: the index results in a value of zero if both sets share no common elements and a value of one if both sets are identical. A summary of our results is shown in Table 2. Only for comparison to the case of SRAM PUFs we also give numbers for the classic fractional Hamming-distance based metrics.

Uniqueness. In order to evaluate the uniqueness of the PUF, we consider the set of indices of DRAM cells that flipped due to decay among different PUFs. In particular, based on two measurements m_1, m_2 that were obtained from two different logical PUFs for the same decay time t , we construct two corresponding sets v_1 and v_2 that store the indices of the flipped cells. Uniqueness is measured by the Jaccard index $J_{inter}(v_1, v_2)$:

$$J_{inter}(v_1, v_2) = \frac{|v_1 \cap v_2|}{|v_1 \cup v_2|}. \tag{1}$$

For an ideal PUF, the value of $J_{inter}(v_1, v_2)$ should be close to zero, indicating that two logical PUFs rarely share flipped bits. Indeed, as Table 2 shows, our DRAM PUFs depict an almost perfect behavior with the Intel Galileo having a maximum of $J_{inter} = 0.0072$ at $t = 360s$. The PandaBoard shows larger values with a maximum value of 0.0405 at $t = 300s$ which, however, is still close to the optimal value of zero.

Robustness. In order to quantify the inherent noise in the PUF measurements and consequently PUF robustness, we computed the Jaccard index $J_{intra}(v_1, v_2)$ between two sets containing the indices of flipped bits in two measurements of

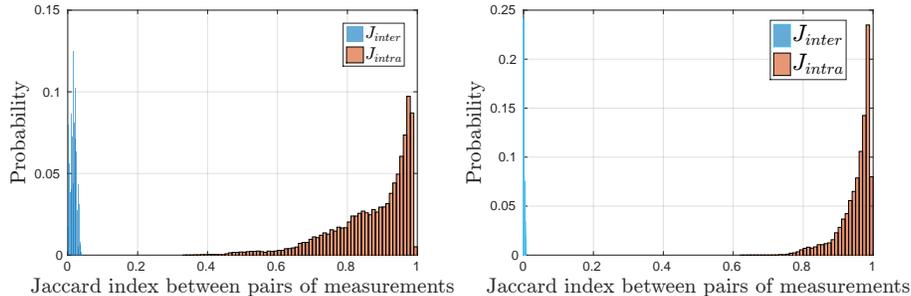


Fig. 3: Distribution of J_{intra} and J_{inter} values for (left) the PandaBoard and (right) the Intel Galileo.

the *same* logical PUF at identical decay times. An ideal PUF should show values close to one, indicating that responses are stable.

Figure 3 displays the distributions of $J_{intra}(v_1, v_2)$ and $J_{inter}(v_1, v_2)$ of all measurements, corresponding to identical decay times, for both device types. A clear divide between the two distributions indicates that individual devices can be distinguished perfectly, while the PUF response is stable over subsequent measurements.

Again, for comparison, we also provide data on the fractional intra-Hamming Distance (intra-HD) in Table 2, i.e., the Hamming distance between subsequent measurements. In comparison to SRAM PUFs, the Hamming distance values are much smaller due to the lower number of bit flips within the DRAM PUF. Nevertheless, except for one case, also the minimum inter-HD is multiply larger than the maximum intra-HD, indicating close to perfect separability.

Entropy. In order to generate cryptographic keys from the PUF response, PUF measurements must exhibit sufficient entropy. We estimate the Shannon entropy of DRAM PUF responses as follows. We again consider the set v of indices of DRAM cells that flipped after time t . Denote with k the cardinality of v and with N the total number of DRAM cells. Assuming that the flipped bits are distributed uniformly, as confirmed by our experiments, the probability of observing one set v is: $P(v) = 1/\binom{N}{k}$. The Shannon entropy of the DRAM PUF for a given decay time can thus be calculated using

$$H_t = \log_2 \binom{N}{k}. \quad (2)$$

Note, that simply observing the number of bits decaying after time t has elapsed, is not sufficient for determining k , as the bit decay will be due to two effects: (i) short-term noise that must be corrected and (ii) stable long-term decay characteristics. In order to approximate k , indicating the stable PUF characteristics, multiple measurements for a single PUF can be averaged in order to eliminate the noise component. Table 2 lists the fractional entropy H_t/N computed this way. We observe that the entropy is significantly bigger on the

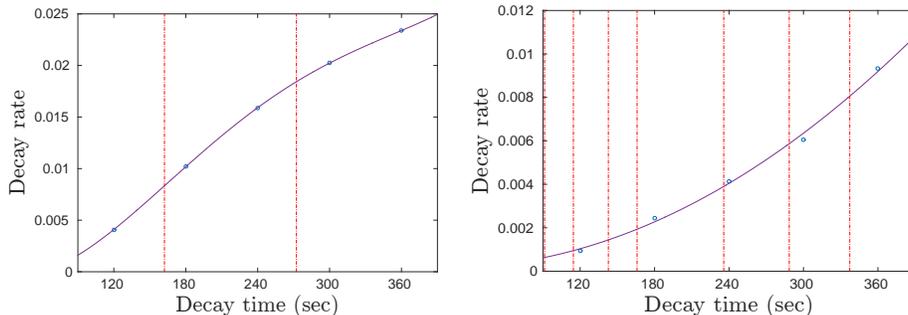


Fig. 4: Time-dependency of decay rate for DRAM modules on the (left) PandaBoard and the (right) Intel Galileo at room temperature.

PandaBoard, indicating more bit flips than on the Intel Galileo. This is most likely due to the different technologies used to implement DRAM cells.

It is noteworthy to compare the entropy that can be extracted from different PUF implementations. While SRAM PUFs usually show min-entropy values of around 0.7 – 0.9 bits per cell, the entropy of the proposed DRAM PUF is one order of magnitude smaller. This can be explained as follows: whilst within SRAM the majority of cells have a unique startup pattern, in case of DRAM only some cells will flip during the observed decay time. However, this lower entropy can be easily compensated by the magnitudes higher (usually a thousand times) amount of DRAM cells available.

Decay Dependency on Time and Temperature. Figure 4 shows the decay rate as a function of decay time for both the PandaBoard and Intel Galileo. All measurements were taken at ambient room temperature with DRAM chips operating at around 40°C. Every data point shows the average of all logical PUFs. We see that the decay rate significantly increases with time on the Galileo. The PandaBoard shows an s-like decay that has a steep beginning and saturates towards $t = 360$ s.

This plot allows us to estimate the number of time-dependent challenges that a logical PUF can provide. In order to allow for unique identification at any given decay time, the set of decay times t_1, t_2, \dots, t_n must be chosen such that the corresponding measurements show a minimum number of *new* bits flips, referred to as ϵ_{bits} , which is greater than the inherent noise. Given ϵ_{bits} , the set of viable decay times (and thus the challenges of a logical PUF) can be chosen accordingly. We used the maximum noise level previously observed for each respective decay time t in order to get a conservative approximation of the maximum number of challenges per logical PUF. We experimentally determined the maximum number of decay times to be $n = 7$ for the Intel Galileo and $n = 2$ for the PandaBoard. The number assumes a maximum decay time $t_n \leq 360$ s and possible challenges are indicated by vertical red lines in Figure 4. The smaller number for the PandaBoard is mainly due to higher noise. In particular, we

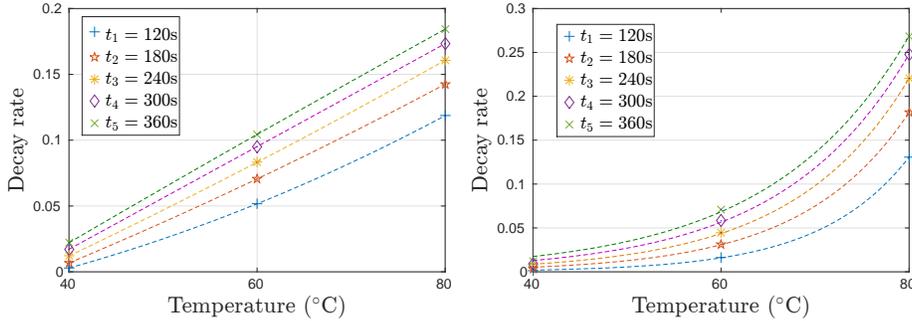


Fig. 5: Relation between the temperature and the decay rate measured on (left) the PandaBoard and (right) the Intel Galileo.

observe that for the PandaBoard the J_{intra} values can be comparably low, i.e. $J_{intra} = 0.3484$ at $t = 360s$. However, given J_{inter} is a magnitude different from J_{intra} , unique identification ability is preserved.

A second factor influencing the decay rate of DRAM cells is temperature. In Figure 5 we show the dependency between temperature and decay rate for DRAM modules on the Intel Galileo and the PandaBoard. In order to control the temperature, we used a metal ceramic heater to heat the surface of DRAM modules to the desired temperature and took the measurements.

Although temperature affects the decay rate significantly, it does not change the decay characteristics much; instead, it affects decay time: We observed that by using a carefully chosen smaller decay time $t'_{T'} < t$ at a larger temperature $T' > T$, the same PUF response can be obtained as with decay time t at temperature T . In our experiments, we derive the following dependency for the Intel Galileo boards:

$$t'_{T'} = t * e^{-0.0662*(T'-T)}. \quad (3)$$

Hence, if the PUF is evaluated at a different temperature than during enrollment, this can be compensated through adapting the decay time according to Equation (3). In order to support this statement, we calculated the noise J_{intra} between an enrollment measurement at room temperature ($40^{\circ}C$) and a measurement taken at a different temperature by adjusting decay time. For this purpose, we created reference measurements at room temperature with decay times $t_x = \{120s, 180s, 240s, 300s, 360s\}$. In a next step, we used equivalent decay times $t'_{T'}$ that correspond to temperatures $T' = \{40^{\circ}C, 50^{\circ}C, 60^{\circ}C\}$ and measured the PUF accordingly. As shown in Figure 6, for all measurements, J_{intra} lies within the usual noise level. Thus, differences in temperature can be accommodated by adjusting decay time accordingly.

Stability over Time. During extended lifetime of devices, DRAM aging effects will begin to take place. Existing work on SRAM PUFs has explored aging ef-

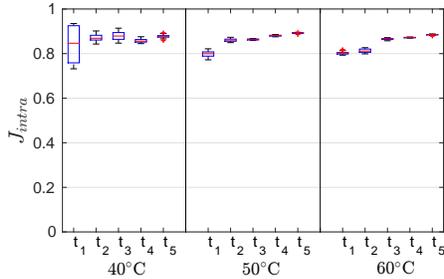


Fig. 6: J_{intra} values (i.e., similarity) of enrollment measurements taken at room temperature and measurements at higher temperatures $T' = \{40^\circ\text{C}, 50^\circ\text{C}, 60^\circ\text{C}\}$, with adjusted decay times t' .

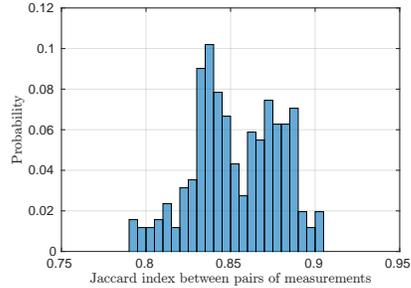


Fig. 7: Distribution of J_{intra} of enrollment, reconstruction measurements pairs taken from the same logical PUF instances on Intel Galileo over four months.

fects [38, 25, 23, 32]. We are aware of limited work on aging-related effects in DRAM cells with regard to security [36]. Figure 7 shows the histogram of J_{intra} values for measurements of an Intel Galileo, taken 4 months apart. Three logic PUFs were measured, and results are combined. Note that the measurements also include the noise introduced by temperature changes in our lab. J_{intra} values were computed of measurement pairs that comprise an enrollment and a reconstruction measurement each. The values 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 Lightweight Protocols for Device Authentication and Secure Channel Establishment

In this section we propose two novel PUF-based protocols that draw their security from the time-dependent decay characteristics of a DRAM PUF instance when queried at different decay times. Both protocols involve two parties, a client \mathcal{C} and server \mathcal{S} . Whilst the first protocol authenticates \mathcal{C} towards an honest \mathcal{S} , the second protocol establishes a secure channel between \mathcal{C} and \mathcal{S} . The protocols leverage PUF instances extracted from DRAM modules and thus require \mathcal{C} to own a device \mathcal{D} that implements a DRAM PUF during the course of the protocol. For the sake of clarity, we will refer to the PUF instance on the client’s device as \mathcal{C} itself. Further, we omit the full specification parameters of the logical PUF instance to be queried. Instead of stating all parameters ($addr, size$) in every protocol, we refer to one logical PUF instance as id .

Adversary and Threat Model. Our adversary model for the protocols considers a passive attacker, who is able to observe the network traffic between client and server and who can capture transmitted messages, in particular previous PUF measurements that were sent by the client. Furthermore, we consider the Fuzzy

Extractor construction, in particular the ECC parameters as well as the Helper Data, to be public and thus known by the attacker.

Enrollment. An enrollment phase precedes both protocols, which is assumed to be conducted at a trusted party \mathcal{SYS} , such as a manufacturer or a system integrator. For each logical PUF instance, during the enrollment phase, \mathcal{SYS} queries the PUF n times in order to get a set of measurements $\mathcal{M} = \{m_{id,0}, m_{id,1}, \dots, m_{id,n}\}$ at a defined set of decay times $\mathcal{T} = \{t_0, t_1, \dots, t_n\}$, i.e., $m_{id,x} = PUF(id, t_x)$. Decay times t_0, t_1, \dots, t_n are carefully chosen such that $t_0 < t_1 < \dots < t_n$ and for every tuple of subsequent decay times the number of newly introduced bit flips in PUF measurements is always greater than a security parameter ϵ_{bits} . The parameter ϵ_{bits} can be changed to adjust security and usability of the protocol (see the end of this section).

To generate keys for the secure channel establishment protocol, \mathcal{SYS} chooses a set $\mathcal{K} = \{k_{id,0}, k_{id,1}, \dots, k_{id,n}\}$ containing uniformly distributed keys and uses a Fuzzy Extractor to create a set of Helper Data $\mathcal{W} = \{w_{id,0}, w_{id,1}, \dots, w_{id,n}\}$, such that $(k_{id,x}, w_{id,x}) = GEN(m_{id,x})$, where $GEN(\cdot)$ denotes the generation function of the Fuzzy Extractor. While the current Fuzzy Extractor constructions [5, 24] might leak entropy from the helper data in case of biased PUF, we assume there is a construction tailored for DRAM PUFs. Eventually, \mathcal{T} , \mathcal{M} , \mathcal{W} and \mathcal{K} will be given to \mathcal{S} , whilst the device will be handed to \mathcal{C} in a secure manner.

Device Authentication. In order to authenticate the client \mathcal{C} towards an honest server \mathcal{S} , the server chooses the smallest decay time t_x not previously used for logical PUF id in a run of the authentication protocol. Next, \mathcal{S} transmits id and t_x to \mathcal{C} , who uses it as input to his or her PUF to retrieve a measurement $m'_{id,x}$, which is sent back to \mathcal{S} . \mathcal{S} checks if $m'_{id,x}$ is close enough to a stored measurement $m_{id,x}$. This is done by checking whether the Jaccard index of $m'_{id,x}$ and $m_{id,x}$ is larger than a given threshold ϵ_{auth} , defined based on the noise of measurement m_x . This authentication protocol is depicted on the left side of Figure 8. Note that for subsequent authentication trials, decay times are monotonically increasing.

The authentication is designed to be lightweight for the client in terms of computational overhead and memory footprint. It does not require \mathcal{C} to store any long-term Helper Data or perform expensive decoding that is usually part of the key reconstruction process performed by classical Fuzzy Extractors. This is especially useful in the context of highly resource-constrained low-cost devices that have to be authenticated towards a server repeatedly.

Secure Channel Establishment. Using similar ideas, a secure channel can be established between \mathcal{C} and \mathcal{S} , see Figure 8 (right side). Again, \mathcal{S} sends the smallest, not previously used decay time t_x for logical PUF id , this time along with the corresponding Helper Data $w_{id,x}$. The client evaluates his PUF instance id using t_x in order to retrieve $m'_{id,x}$, which is used in combination with $w_{id,x}$ to reconstruct $k'_{id,x} = REC(m'_{id,x}, w_{id,x})$. If the measurement m'_x was obtained using the correct logical PUF, the resulting key $k'_{id,x}$ will be identical to the key $k_{id,x}$ stored at the server in \mathcal{K} . Thus, both parties \mathcal{C} and \mathcal{S} will share the same key.

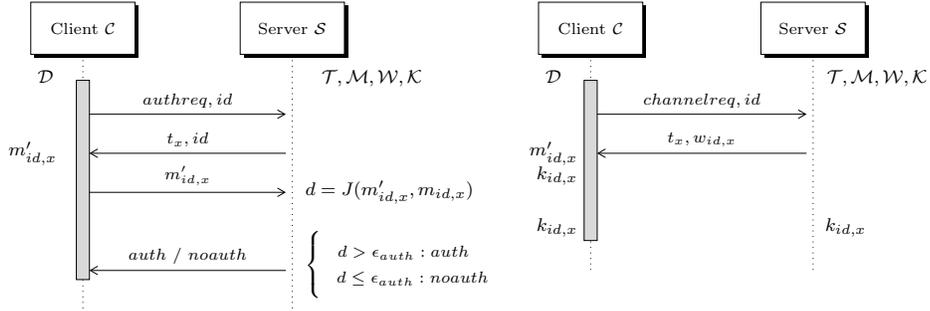


Fig. 8: Sequence diagram of (left) the device authentication protocol and (right) the secure channel establishment protocol.

In contrast to the authentication protocol depicted above, secure channel establishment is less lightweight, as it requires the evaluation of the Fuzzy Extractor on the client.

On The Choice of Security Parameters. Following our threat model, the attacker can obtain all previously used PUF measurements m_x by eavesdropping the authentication protocol, the security of the protocol is inherently based on the number of *newly* flipped bits ϵ_{bits} that emerge between measurements for subsequent decay times t_x, t_{x+1} . The value of ϵ_{bits} , and correspondingly the decay times, must be chosen in a way that a new PUF measurement has enough new entropy, even if the attacker knows the previous measurements.

In order to forge authentication or to derive the session key, the attacker has to guess the PUF measurement corresponding to the next unused decay time. In order to do so, the best strategy, without the knowledge of the physical PUF characteristics, is to randomly guess where the bit flips in the subsequent measurement will occur, knowing the previous measurement m_x . Suppose that the attacker guesses M new bit flips. Then, he can use these M flips together with the bit flips in the old response m_x as his guess of the new PUF measurement m_{x+1} . In the next paragraphs we estimate the probability of success for such a strategy. This allows us to fix the security parameter ϵ_{bits} , such that the probability of a successful guess is small.

We estimate the success probability as follows. The space of potential new bit flips is of size N , which is the number of bits that did not flip in m_x . Out of the remaining N bits, the attacker can guess M bits and combine them with m_x in order to generate m'_{x+1} . Note that the attacker does not need to guess the exact pattern m_{x+1} . Instead, the attacker will be successful, if the guessed measurement m'_{x+1} is a noisy version of the true measurement, i.e., m'_{x+1} lies within the error-correction bounds of the Fuzzy Extractor.

The probability that an attacker guesses M random bits and l of them happen to be real new bit flips of the subsequent measurement is $\frac{\binom{\epsilon_{bits}}{l} \binom{N-\epsilon_{bits}}{M-l}}{\binom{N}{M}}$.

Note that in this case, the Jaccard index of the attacker’s guess and the true measurement is $J(m'_{x+1}, m_{x+1}) = \frac{l+|m_x|}{\epsilon_{bits}+M-l+|m_x|}$, where $|m_x|$ is the number of bit flips in the previous measurement m_x . Assuming the authentication and key generation is successful if $J(m'_{x+1}, m_{x+1}) > \Delta$, the attacker will only be successful if l is greater than $\frac{(\epsilon_{bits}+M)*\Delta}{1+\Delta} - \frac{|m_x|*(1-\Delta)}{1+\Delta}$. Thus, the probability for an attacker to make a successful guess is:

$$P_M = \sum_{l=\lceil \frac{(\epsilon_{bits}+M)*\Delta}{1+\Delta} - \frac{|m_x|*(1-\Delta)}{1+\Delta} \rceil}^{\epsilon_{bits}} \frac{\binom{\epsilon_{bits}}{l} \binom{N-\epsilon_{bits}}{M-l}}{\binom{N}{M}}. \quad (4)$$

The attacker can chose any M , which will maximize the success probability P_M . If N is large and M is between $M_{min} = \epsilon_{bits} * \Delta - (1 - \Delta)|m_x|$ and $M_{max} = \frac{\epsilon_{bits}+(1-\Delta)*|m_x|}{\Delta}$, P_M is monotonically decreasing with M .⁸ Hence, the attacker can choose $M = M_{min}$ to maximize the success probability.

In order to provide 128-bit security, $P = \max_M \{P_M\}$ must be smaller than 2^{-128} . Given Formula 4 and PUF characteristics, one can fix N and Δ , then derive ϵ_{bits} for different $|m_x|$, and subsequently estimate the feasible decay time. For the Intel Galileo, as a conservative estimation, the space of potential new bit flips is of size $N = 30KB$ (assuming that out of a 32KB logical PUF, less than 2KB are flipping in m_x), and the threshold is $\Delta = 0.6$. To set $|m_0|$, a point where the decay is larger than the noise should be found. To be conservative, minimum max intra-HD is used as a reference for $|m_0|$.⁹ Hence, we set $|m_0| = 80$, and then we can get $\epsilon_{bits1} = 73$, and $|m_1| = |m_0| + \epsilon_{bits2} = 153$. Then with $|m_1|$, we can get $\epsilon_{bits2} = 122$ and thus $|m_2| = 275$, etc. Consequently, in the Galileo, a 32KB logical PUF can provide 7 challenges, each with the decay time shorter than 360s.

6 Open Research Topics

This novel work on run-time accessible DRAM PUFs still leaves a number of open research issues and questions that need to be addressed. This creates opportunities for the community to refine and further improve the concept of DRAM PUFs.

Temperature dependency of the DRAM cell decay allows physical attackers to control the decay rate by adjusting the ambient temperature. For example, heating a DRAM chip may “speed up” the decay rate, shortening the time needed for an attacker to observe certain bits flip. Further investigation on the temperature dependence is needed and counter-measures need to be developed to thwart such attacks.

Voltage dependency of the DRAM cell decay was not considered in this paper, as commodity devices usually give no control over DRAM voltages. However, voltage dependency could be another viable characteristic used for the run-time

⁸ $P_M > 0$ when M is between $\epsilon_{bits} * \Delta - (1 - \Delta)|m_x|$ and $\frac{\epsilon_{bits}+(1-\Delta)*|m_x|}{\Delta}$.

⁹ If the PUF characteristic is better understood for $t < 120s$, a smaller $|m_0|$ may be chosen.

accessible DRAM PUFs, if future commodity devices provide interfaces that allow for fine-grained control of DRAM voltages.

Readout time of the DRAM PUFs is in the order of minutes. This can be seen as a disadvantage, although in many cases it can be compensated by the advantage of being able to access the DRAM PUFs at run-time. Use cases that allow for such relatively long readout times need to be better understood. At the same time, improving the readout time is critical in order to broaden the applicability of DRAM PUFs.

Security assumptions, e.g., the trusted firmware and the operating system, may be considered as too strong. While these are also required for the other PUFs in commodity devices, one may look for solutions requiring a smaller trusted computing base.

Fuzzy Extractor constructions are needed that are either specifically tailored towards heavily biased PUF responses, found in decay-based DRAM PUFs, or that use the introduced Jaccard distances. Classic Fuzzy Extractors are based on Hamming distance-related metrics and are not secure for heavily biased PUFs. Thus, new constructions for biased PUFs, such as [24, 39], should be developed.

7 Conclusion

In this work we presented intrinsic PUFs that can be extracted from Dynamic Random-Access Memory (DRAM) 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, as well as stability over period of at least few months. Moreover, in contrast to existing DRAM and SRAM PUFs, we demonstrate a system model that is able to query the PUF instance directly during run-time using a Linux kernel module, based on the ideas of *selective DRAM refresh* and *memory ballooning*. We further presented protocols for device authentication and identification that draw their 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 use of decay time t as part of the challenge. Consequently, our work presents a new alternative for device authentication by leveraging DRAM in commodity devices.

8 Acknowledgements

This work has been co-funded by the DFG as part of project P3 within the CRC 1119 CROSSING. This work was also partly funded by CASED. The authors would like to thank Kevin Ryan and Ethan Weinberger for their help with building the heater setup used in the experiments, and Intel for donating the Intel Galileo boards used in this work. The authors would also like to thank anonymous CHES reviewers, and especially our shepherd, Roel Maes, for numerous suggestions and guidance in making the final version of this paper.

References

1. Hacking DefCon 23's IoT Village Samsung fridge, <https://www.pentestpartners.com/blog/hacking-defcon-23s-iot-village-samsung-fridge/>, accessed Feb. 2016
2. Armknecht, F., Maes, R., Sadeghi, A.R., Sunar, B., Tuyls, P.: Memory leakage-resilient encryption based on physically unclonable functions. In: *Towards Hardware-Intrinsic Security*, pp. 135–164. Springer (2010)
3. Bacha, A., Teodorescu, R.: Authenticache: harnessing cache ECC for system authentication. In: *Proceedings of International Symposium on Microarchitecture*. pp. 128–140. ACM (2015)
4. Batra, P., Skordas, S., LaTulipe, D., Winstel, K., Kothandaraman, C., Himmel, B., Maier, G., He, B., Gamage, D.W., Golz, J., et al.: Three-dimensional wafer stacking using Cu TSV integrated with 45 nm high performance SOI-CMOS embedded DRAM technology. *Journal of Low Power Electronics and Applications* pp. 77–89 (2014)
5. Dodis, Y., Reyzin, L., Smith, A.: Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In: *Advances in Cryptology – Eurocrypt*. pp. 523–540. Springer (2004)
6. Foster, I., Prudhomme, A., Koscher, K., Savage, S.: Fast and vulnerable: a story of telematic failures. In: *USENIX Workshop on Offensive Technologies* (2015)
7. Gassend, B., Clarke, D., Van Dijk, M., Devadas, S.: Delay-based circuit authentication and applications. In: *Proceedings of the ACM Symposium on Applied Computing*. pp. 294–301. ACM (2003)
8. Greenberg, A.: Hackers remotely kill a jeep on the highway—with me in it. *Wired*, 21 July (2015)
9. Guajardo, J., Kumar, S.S., Schrijen, G.J., Tuyls, P.: FPGA intrinsic PUFs and their use for IP protection. Springer (2007)
10. Guajardo, J., Kumar, S.S., Schrijen, G.J., Tuyls, P.: Brand and IP protection with physical unclonable functions. In: *IEEE International Symposium on Circuits and Systems*. pp. 3186–3189 (2008)
11. Hashemian, M.S., Singh, B., Wolff, F., Weyer, D., Clay, S., Papachristou, C.: A robust authentication methodology using physically unclonable functions in DRAM arrays. In: *Proceedings of the Design, Automation & Test in Europe Conference*. pp. 647–652 (2015)
12. Hernandez, G., Arias, O., Buentello, D., Jin, Y.: Smart nest thermostat: A smart spy in your home. *Black Hat USA* (2014)
13. Jaccard, P.: *Etude comparative de la distribution florale dans une portion des Alpes et du Jura*. Impr. Corbaz (1901)
14. Katzenbeisser, S., Kocabaş, Ü., Rožić, V., Sadeghi, A.R., Verbauwhede, I., Wachsmann, C.: PUFs: Myth, fact or busted? A security evaluation of physically unclonable functions (PUFs) cast in silicon. In: *Cryptographic Hardware and Embedded Systems*, pp. 283–301. Springer (2012)
15. Keeth, B.: *DRAM circuit design: fundamental and high-speed topics*. John Wiley & Sons (2008)
16. Keller, C., Gurkaynak, F., Kaeslin, H., Felber, N.: Dynamic memory-based physically unclonable function for the generation of unique identifiers and true random numbers. In: *IEEE International Symposium on Circuits and Systems*. pp. 2740–2743. IEEE (2014)

17. Kim, Y., Daly, R., Kim, J., Fallin, C., Lee, J.H., Lee, D., Wilkerson, C., Lai, K., Mutlu, O.: Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In: ACM SIGARCH Computer Architecture News. pp. 361–372 (2014)
18. Kocabaş, Ü., Peter, A., Katzenbeisser, S., Sadeghi, A.R.: Converse PUF-based authentication. Springer (2012)
19. Kohnhäuser, F., Schaller, A., Katzenbeisser, S.: PUF-Based Software Protection for Low-End Embedded Devices. In: Trust and Trustworthy Computing, pp. 3–21. Springer (2015)
20. Kong, J., Koushanfar, F., Pendyala, P.K., Sadeghi, A.R., Wachsmann, C.: PU-Fatt: Embedded platform attestation based on novel processor-based PUFs. In: ACM/EDAC/IEEE Design Automation Conference. pp. 1–6 (2014)
21. Liu, J., Jaiyen, B., Kim, Y., Wilkerson, C., Mutlu, O.: An experimental study of data retention behavior in modern DRAM devices: Implications for retention time profiling mechanisms. In: ACM SIGARCH Computer Architecture News. pp. 60–71 (2013)
22. Liu, W., Zhang, Z., Li, M., Liu, Z.: A Trustworthy Key Generation Prototype Based on DDR3 PUF for Wireless Sensor Networks. *Sensors* pp. 11542–11556 (2014)
23. Maes, R., van der Leest, V.: Countering the effects of silicon aging on SRAM PUFs. In: IEEE International Symposium on Hardware-Oriented Security and Trust. pp. 148–153 (2014)
24. Maes, R., van der Leest, V., van der Sluis, E., Willems, F.: Secure key generation from biased pufs. In: Cryptographic Hardware and Embedded Systems—CHES 2015, pp. 517–534. Springer (2015)
25. Maes, R., Rožić, V., Verbauwhede, I., Koeberl, P., Van der Sluis, E., Van der Leest, V.: Experimental evaluation of Physically Unclonable Functions in 65 nm CMOS. In: Proceedings of the ESSCIRC. pp. 486–489 (2012)
26. Phone as a Token - turn your phone into an authentication token, <https://www.intrinsic-id.com/technology/phone-as-a-token/>, accessed Feb. 2016
27. Rahmati, A., Hicks, M., Holcomb, D.E., Fu, K.: Probable cause: the deanonymizing effects of approximate DRAM. In: Proceedings of the International Symposium on Computer Architecture. pp. 604–615 (2015)
28. Rosenblatt, S., Chellappa, S., Cestero, A., Robson, N., Kirihata, T., Iyer, S.S.: A Self-Authenticating Chip Architecture Using an Intrinsic Fingerprint of Embedded DRAM. *IEEE Journal of Solid-State Circuits* pp. 2934–2943 (2013)
29. Rosenblatt, S., Fainstein, D., Cestero, A., Safran, J., Robson, N., Kirihata, T., Iyer, S.S.: Field tolerant dynamic intrinsic chip ID using 32 nm high-K/metal gate SOI embedded DRAM. *IEEE Journal of Solid-State Circuits* pp. 940–947 (2013)
30. Rührmair, U., Sölter, J., Schnke, F.: On the Foundations of Physical Unclonable Functions. *IACR Cryptology ePrint Archive* 2009, 277 (2009)
31. Schaller, A., Arul, T., van der Leest, V., Katzenbeisser, S.: Lightweight Anti-counterfeiting Solution for Low-End Commodity Hardware Using Inherent PUFs. In: Trust and Trustworthy Computing, pp. 83–100. Springer (2014)
32. Schaller, A., Škorić, B., Katzenbeisser, S.: On the systematic drift of physically unclonable functions due to aging. In: Proceedings of the International Workshop on Trustworthy Embedded Devices. pp. 15–20. ACM (2015)
33. Scheel, R.A., Tyagi, A.: Characterizing composite user-device touchscreen physical unclonable functions (pufs) for mobile device authentication. In: Proceedings of the International Workshop on Trustworthy Embedded Devices. pp. 3–13. ACM (2015)
34. Schneier, B.: The internet of things is wildly insecure—and often unpatchable. *Wired*, Jan (2014)

35. Schrijen, G.J., van der Leest, V.: Comparative analysis of SRAM memories used as PUF primitives. In: Proceedings of the Conference on Design, Automation and Test in Europe. pp. 1319–1324. EDA Consortium (2012)
36. Schroeder, B., Pinheiro, E., Weber, W.D.: DRAM errors in the wild: a large-scale field study. In: ACM SIGMETRICS Performance Evaluation Review. pp. 193–204 (2009)
37. Schulz, S., Sadeghi, A.R., Wachsmann, C.: Short paper: lightweight remote attestation using physical functions. In: Proceedings of the ACM Conference on Wireless Network Security. pp. 109–114 (2011)
38. Selimis, G., Konijnenburg, M., Ashouei, M., Huisken, J., De Groot, H., Van der Leest, V., Schrijen, G.J., Van Hulst, M., Tuyls, P.: Evaluation of 90nm 6T-SRAM as Physical Unclonable Function for secure key generation in wireless sensor nodes. In: IEEE International Symposium on Circuits and Systems. pp. 567–570 (2011)
39. Skoric, B.: A trivial debiasing scheme for helper data systems. Cryptology ePrint Archive, Report 2016/241 (2016)
40. Suh, G.E., Devadas, S.: Physical unclonable functions for device authentication and secret key generation. In: Proceedings of the Design Automation Conference. pp. 9–14 (2007)
41. Tehranipoor, F., Karimina, N., Xiao, K., Chandy, J.: DRAM based Intrinsic Physical Unclonable Functions for System Level Security. In: Proceedings of the Great Lakes Symposium on VLSI. pp. 15–20 (2015)
42. Intrinsic-ID to Showcase TrustedSensor IoT Security Solution at InvenSense Developers Conference, <https://www.intrinsic-id.com/intrinsic-id-to-showcase-trustedsensor-iot-security-solution-at-invensense-developers-conference/>, accessed Feb. 2016
43. Tuyls, P., Batina, L.: RFID-tags for Anti-Counterfeiting. In: Topics in Cryptology, pp. 115–131. Springer (2006)
44. Tuyls, P., Schrijen, G.J., Willems, F., Ignatenko, T., Skoric, B.: Secure key storage with PUFs. Security with Noisy Data—On Private Biometrics, Secure Key Storage and Anti-Counterfeiting pp. 269–292 (2007)
45. Tuyls, P., Škorić, B.: Secret key generation from classical physics: Physical unclonable functions. In: AmIware Hardware Technology Drivers of Ambient Intelligence, pp. 421–447. Springer (2006)
46. Viega, J., Thompson, H.: The State of Embedded-Device Security (Spoiler Alert: It’s Bad). IEEE Security & Privacy pp. 68–70 (2012)
47. Waldspurger, C.A.: Memory Resource Management in VMware ESX Server. ACM SIGOPS Operating Systems Review pp. 181–194 (2002)