Practical DRAM PUFs in Commodity Devices

Wenjie Xiong[1], André Schaller[2], Nikolaos Anagnostopoulos[2],
Muhammad Umair Saleem[2], Sebastian Gabmeyer[2],
Stefan Katzenbeisser[2], and Jakub Szefer[1]

[1] Yale University, New Haven CT 06511, USA,
{wenjie.xiong,jakub.szefer}@yale.edu
[2] Technische Universität Darmstadt and CASED, Darmstadt, Germany
{schaller,anagnostopoulos,gabmeyer,katzenbeisser}@seceng.informatik.tu-darmstadt.de
umairsaleemchaudhry@gmail.com

Abstract. A Physically Unclonable Function (PUF) is a unique and stable physical characteristic of a piece of hardware, due to variations in the fabrication processes. Prior works have demonstrated that PUFs are a promising cryptographic primitive to enable hardware-based device authentication and identification. A diverse number of PUFs have been explored, e.g., delay-based PUFs in dedicated circuits, SRAM-based PUFs in commodity hardware, and DRAM-based PUFs in custom FPGA-based setup. This paper is the first to extract and evaluate a DRAM PUFs from commodity off-the-shelf hardware and to provide a practical solution to query the PUF during a Linux system run-time, not just at startup. DRAM instances are traditionally larger compared to SRAM and thus provide an increased challenge-response space that makes them attractive. Lightweight protocols for device authentication and secure channel establishment are proposed, that exploit this large challenge-response space of the DRAM PUFs and the time-dependent decay of DRAM cells. Intrinsic DRAM PUF characteristics are evaluated based on commodity hardware using custom Linux kernel module and also firmware code.

1 Introduction

Continued miniaturization and cost reductions of processors and System-on-a-Chip (SoC) designs have enabled creation of almost ubiquitous smart devices, from smart thermostats [1, 16] and smart refrigerators [2] to smart phones and embedded car entertainment systems [12, 7]. While there are numerous advantages to the proliferations of such smart devices, the increasing number of them creates new security vulnerabilities and concerns. Many news articles have pointed out that such devices often lack the implementation of sufficient security mechanisms [37, 50]. Critical challenges in securing these devices are to provide robust device authentication and identification mechanisms as well as means storing long-term cryptographic keys in a secure manner that minimized the chances of their illegitimate extraction.

A classic approach 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 [3]. In order to address these issues, researchers have proposed Physically Unclonable Functions (PUFs). PUFs enable to use unique behavior of a device due to manufacturing variation as a hardware-based fingerprint. A PUF measurement[3] is extremely difficult to replicate, even by the manufacturer themselves. So far, most types of PUFs (such as arbiter PUFs [42]) required addition of dedicated circuits to the device and thus increased manufacturing costs and hardware complexity. Consequently, there is great interest in so-called intrinsic PUFs [13], i.e., PUFs that are already inherent in a device.

Intrinsic PUFs are considered as an attractive security anchor, as they provide PUF instances within standard hardware that can be found in Commercial Off-The-Shelf (COTS) devices [45, 28]. The most prominent representative of an intrinsic PUF is a PUF based on Static Random-Access Memory (SRAM) [34, 21, 49, 36], which draws its unique characteristic from the startup values of bistable SRAM memory cells. The cells' individual tendencies to either initialize to zero or one creates a unique, hardware-based fingerprint for a given SRAM module. However, in this case PUF measurements must be extracted during a very early boot stage (before the SRAM is used) and 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 access at run-time was proposed [4]. However, to query the PUF, the supply voltage need to be lowered to induce errors to SRAM cells, requiring special hardware in the processor. Furthermore, SRAM-based PUFs are also considered as "weak" PUFs as they have a limited challenge-response space. Consequently, rather than using such PUFs as part of authentication and identification protocols, they are mainly used within secure key storage solutions [33].

Most recently, PUF-like behavior has been found in Dynamic Random-Access Memory (DRAM) [30]. Current approaches to extract unique DRAM behavior induced by manufacturing variations, rely on the individual charge decay of DRAM cells [18] as well as startup tendencies of DRAM cells [43]. Exploiting those effects, a unique hardware-based fingerprint can be derived from the DRAM PUF response in order to robustly identify the memory module. DRAM is an integral part of various types of commodity devices and can be found in many "smart" devices, e.g. smartphones or smart thermostats. Recent use of embedded DRAM (eDRAM) [52, 11, 31, 5] in low-cost microprocessors (MCUs) will further increase the availability of DRAM as part of mobile and embedded computing platforms. Thus, DRAM in COTS hardware is a good candidate for an intrinsic PUF.

Decay-based DRAM PUFs can allow for repeated access at run time, which allows for interfacing with the operating system (OS) and user space applications. Furthermore, the capacity of DRAM is magnitudes larger than SRAM (usually

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

by a factor of 1000 to 10000), increasing the number of challenges and resulting responses of the PUFs. This allows for drawing more bits in order to derive more cryptographic keys and subsequently allows to use DRAM-based PUFs as part of cryptographic applications and protocols that are not limited to secure key storage scenarios. However, the current state of art to use DRAM-based PUFs employs custom hardware, FPGA-based platforms in particular [43, 18].

In contrast, this paper is the first to extract and evaluate a DRAM PUFs from commodity off-the-shelf hardware and to provide a practical solution to query the PUF during a Linux system run-time, not just at startup. Especially, our constructed DRAM PUF works by: initializing region of memory to predefined value, disabling memory refresh, waiting for certain time, reading back memory contents (including the bits that decayed) as PUF measurement. By disabling memory refresh, bit value stored in the DRAM cells decays, which exhibits unique characteristics in PUF response. Using the decay time as part of the PUF challenge allows for increasing the number of challenge/response pairs. Subsequently, the increased number of extractable keying material enables the usage of DRAM-based PUFs as part of device authentication and identification protocols.

Contributions: To the best of our knowledge, this is the first work to present an approach to extract DRAM PUF instances from COTS devices and to provide a system-level solution for querying the DRAM PUF at while a Linux OS is running on same hardware and is actively using DRAM chip where the PUF is located. Our contributions are as follows:

- We evaluate DRAM-based PUF instances extracted from unmodified, commodity devices including the PandaBoard and the Intel Galileo Platform and show that they exhibit useful PUF characteristics.
- We present two approaches for accessing the PUF that allow for querying the PUF i) at device startup, using a customized firmware and ii) during run-time, with the help of a Linux kernel module.
- We show a novel solution that allows for accessing the DRAM-based PUF in a Linux-based operating system, based on two techniques, namely *selective DRAM refresh* and *memory ballooning*. Our evaluation proves the practicability of the proposed approach.
- Finally, we propose to exploit the time-dependent unique decay characteristics of DRAM cells in order to expand the number of challenge/response pairs and further design lightweight protocols that allow for device authentication and identification.

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 as arrays and are connected by a horizontal word-line connecting each row of the array as well as bit-line connecting cells in the same column. All bit-lines are connected to

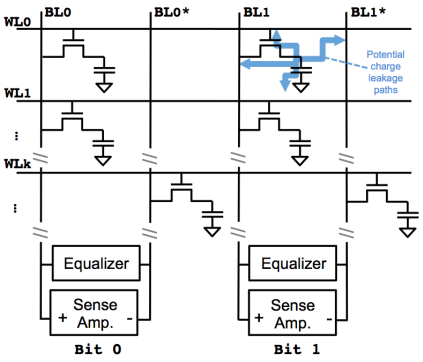

Fig. 1: A single DRAM cell consists of a capacitor, transistor, word-line (WL), bit-lines (BL); arrows indicate possible leakage paths for dissipation of charges; DRAM array also has true cell and anti-cell connected to BL and BL* respectively.

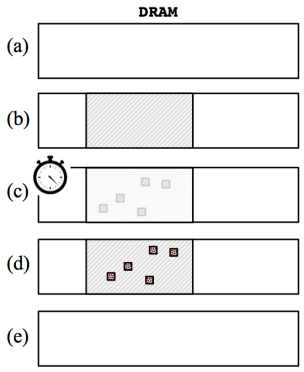

Fig. 2: Illustration of five steps of accessing and reading a DRAM PUF. Only during PUF measurement (b) - (d) is memory associated with the PUF not usable, otherwise it is usable at all times during run-time.

a so-called sense-amplifier that senses low power signals from the bit-lines and to amplify the small voltage 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 the supply voltage $V_{DD}/2$, and then the word-line is *enabled*, connecting every capacitor in that row with its bit-line. A sense amplifier will then drive the bit-line to V_{DD} or 0V, depending the charge in the capacitor. The amplifiers are usually shared by two bit-lines [17], of which only one can be accessed at the same time. This structure make the two bit-lines complementary, which results in two kinds of cells: true cells and anti-cells. True cells store value 1 as V_{DD} and 0 as 0V on the capacitor, whilst anti-cells store value 0 as V_{DD} and 1 as 0V. So even though the charge on the capacitor can only leak, some cells decay to 0, others to 1.

2.1 PUFs in DRAM

The capacitor used to stores data bits leaks the charge over time. Therefore, it has to be periodically refreshed to maintain the stored value in the cell during normal operation. The hardware memory controller takes care of periodic refresh, whose interval is defined by the vendor and is usually 32ms or 64ms. We will exploit this decay characteristic to use the DRAM as a PUF. In particular, DRAM PUFs exploit manufacturing variations among DRAM cells as some cells decay faster then others. After a certain decay time enough charge has leaked from a cell such that the stored logical bit will flip. As the decay time is unique for individual cells, as well as the positions of the entirety of the flipped bits, the DRAM "pattern" of flipped bits for a given decay time t can serve as PUF. We also introduce the concept of logical DRAM PUF, which consists of a memory region within a DRAM module within which the cell decay characteristics are used for PUF measurements. The total DRAM memory can be divided into multiple logical PUFs. For a particular DRAM, each logical PUF is determined

by a number of parameters: i) *addr*: the start address of the logical PUF, ii) *size*: its size, iii) *initval*: the initial values that are written to the memory cells comprising the PUF, and iv) t: the decay time after which the memory contents are read.

In order to derive a cryptographic key from the PUF response, the size *size* of the measured memory region has to be set properly to be able to gather enough entropy for key generation (see Section 5). Furthermore, *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 initial value in that cell is also 0, PUF effect cannot be observed. Effectively, different *initval* values can be used to select which cells within the logical PUF will be observed.

The process of exploiting the unique decay behavior of DRAM cells in order to extract a PUF measurement is summarized in Figure 2. The starting point (a) comprises the DRAM module configured for ordinary use and the memory controller is configured to periodically refresh the cell's contents. In a first step (b), the memory region that corresponds to the logical PUF, which is to be queried, is reserved, e.g. by instructing the kernel to free the respective memory region from stored data and making it unavailable for subsequent write attempts. Furthermore, the refresh for the PUF region is disabled and an initialization value is written to the region. Next, (c) for a given decay time t, which is an essential part of the PUF challenge, the PUF is not accessed to let the cells decay. After t has expired (d), the memory contents are read back in order to extract the PUF measurement. Finally, the normal operating condition of the memory is restored (e) and the memory region is made available to the OS again.

2.2 Practical DRAM PUF Access at Run-time

Deactivating DRAM refresh for PUF access during the operation of any operating system is a non-trivial task: when DRAM refresh cycles are disabled, critical data will start to decay and the system will crash. In our experiments, the Intel Galileo board with Yocto Linux crashes about a minute after the whole DRAM refresh is disabled. In some devices, the DRAM consists of several physical modules, and the refresh of each module can be disabled individually. Then one of the modules can be employed for PUF usage, whose refresh can be disabled for measurement, whilst the others operate under normal condition in order to let the OS store its data. However, many devices (including the ones under test in this research) do not allow for such approach.[4]

Therefore, we present a customized solution, that is based on two techniques dubbed *selective DRAM refresh* and *memory ballooning*. The former technique

[4] Although the test boards do have multiple DRAM modules, they cannot be disabled individually. In particular, on Galileo board, one of DRAM chips is used to provide most significant 8 bits of a memory word, while the other is used for least significant bits of memory word. Disabling one DRAM is then not possible as half of each memory word of would be lost. Also, the PandaBoard has two physical DRAM modules, but the refresh rate is populated simultaneously to modules via a hard-wired signal, which prohibits selectively toggling refresh cycles for individual modules.

allows for selectively refreshing the memory regions occupied by the OS and other critical applications so that it runs normally and does not crash. Memory ballooning, on the other hand, protects memory regions that correspond to logical PUF instances from system access to let respective cells decay.

Selective DRAM Refresh. Memory cells are refreshed implicitly if they are accessed due to a read or write operation. Each read causes the corresponding memory row to be loaded into the DRAM's row buffer, in turn refreshing the whole memory row. When a word line is selected, the sense amplifier drives the bit-lines to either supply voltage V_{DD} or 0. In this way the capacitor is charged back to full V_{DD} or 0. Using this principle, even if refresh cycles of the memory controller are disabled, selective memory locations (at the size of the DRAM row buffer) can be refreshed by reading corresponding memory rows. This functionality can be implemented in a Linux kernel module.

Ballooning System Memory. To query a chosen logical PUF, the DRAM portion given by *addr* and *size* is overwritten by the *initval* and the refresh is deactivated. To protect the OS from accessing the PUF memory regions we use memory ballooning concepts developed for virtual machines [51]. In virtual machines, 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 applications. This idea can be ported to non-virtualized settings. Modifying the standard ballooning scheme allows to specify the specific physical address (*addr*) and size (*size*) of the memory region that will be reserved. 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 memory restored to normal use.

2.3 Comparison of memory-based PUFs

Conventionally, in a *weak PUF*, the number of challenge-response pairs is linear in the size of the PUF. Meanwhile, *strong PUFs* have number of challenge-response pairs that is exponential in size of the PUF [32]. Existing memory-based PUFs fall in the weak PUF regime. The presented DRAM PUF is also a weak PUF because of the linear relation between DRAM size and the number of possible challenge-response pairs. However, the DRAM has a unique characteristic that each logical DRAM PUF will have different decay characteristics depending on time – thus there is a time parameter t that needs to be considered when querying the PUF. Each logical PUF will give different responses based on the time, i.e., $m_i = PUF(addr, size, initval, t_i)$, and for different decay times t_i, different m_i are observed. In order to obtain a set of $(m_0, m_1, ..., m_n)$ PUF responses for $(t_0, t_1, ..., t_n)$ delays, n measurements need to be taken[5].

[5] Note that it is not possible to get all responses for decay times of t_0 to t_n with only one measurement, because during the measurement reading off PUF data at each $t_0, t_1, ..., t_n$ time is the act of sequentially reading the PUF during t_i, which will cause the memory to be refreshed (as data is read from DRAM's cells into row buffers the cells are re-charged).

Table 1: Size of SRAM and DRAM in commercial devices

System	**Processor**	**SRAM (MB)**	**DRAM (MB)**	**DRAM/SRAM Ratio**
Intel Galileo Gen. 2	Intel Quark X1000	0.02	256	16384
Pandaboard ES	TI OMAP 4460	1.00	1024	1024
iPhone 6s	Apple A9	4.00	2048	512
Nest Thermostat Gen.2	TI AM3707	0.25	512	2048
Average		1.85	2406	1300

Table 2: Comparison of properties of SRAM and DRAM PUFs, our construction is listed in right-most column; best entries are highlighted in bold.

	SRAM startup-based PUF	*SRAM error-based PUF [4]*	*DRAM startup-based PUF*	**DRAM delay-based PUF**
memory size	size of SRAM	number of cache lines (about 1/512 SRAM)	**size of DRAM, 1000 to 10000 larger than SRAM**	
PUF parameters	addr	**addr, voltage**	addr	**addr, time**
run-time access	No	**Yes**	No	**Yes**
need special processor feature	**No**	Yes	**No**	**No**

To exhaustively query a DRAM module and obtain m_i for all logical PUFs for all different time steps, the attacker needs to perform n queries for the whole DRAM module. The process can be sped up by increasing the PUF temperature. On the other hand, 256 MB is a conservative assumption about the DRAM size, as many devices provide 1 GB or more, which is usually a thousand times the size of SRAM in standard hardware, as shown in Table 1. This would require the attacker to have enough data storage and processing ability. This makes DRAM PUF stronger than conventional weak PUF. A comparison of different memory-based PUFs is given in Table 2.

2.4 Novel Use-Cases Enabled by DRAM PUFs

The DRAM-based PUF that uses different decay times (in seconds or minutes) and can be queried at run-time opens a number of new use-cases for authentication and secure channel establishment. Unlike memory PUFs that exploit startup values, run-time access to PUFs allows for active device authentication.

Sensor Authentication and Secure Channel Establishment. In sensor applications, the DRAM PUF can be queried while the device collects data. Many sensors send their sensed values in a periodic manner. The delay time between subsequent transmissions of the sensor readings can be utilized to query the PUF in parallel.

Pre-Authentication and Secure Channel Establishment. Devices which are powered before actual usage, e.g. smart TVs, can pre-authenticate and set up a secure channel before device is in use. For example, a device can run prediction algorithm and predict when a user wants to watch a movie, and query the PUF and authenticate with content provider before hand.

Delayed Authentication. A number of applications work well using delayed authentication, for example e-commerce. If a user places an online order on his or her smartphone, the vendor starts the authentication process. As it takes time to find physical items in a warehouse and prepare for shipping, it is sufficient that the user's device is authenticated after purchase but before items are shipped.

3 Implementation and performance

We implemented and tested our solution on two popular platforms, the PandaBoard ES Revision B3 [9] and the Intel Galileo Gen 2 [8]. The PandaBoard houses a TI OMAP 4460 System-on-Chip (SoC) module that implements 1 GB of DDR2 memory (Elpida EDB8064B1PB-8D-F) in a Package-on-Package (PoP) style, which operates at 1.2V. The Intel Galileo has an Intel Quark SoC X1000 SoC with two 128 MB DDR3 (MT41K128M8) from Micron operating at 1.5V. The two physical DRAM modules are accessed in parallel and located on the PCB next to 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. The purpose of the firmware-based approach was used to run tests in an automated manner. Secondly, we implemented a kernel module-based solution that enables PUF queries during run-time of a Linux operating system. We present implementation details of both approaches in the following.

3.1 Firmware-based PUF Access

In order to obtain a large set of test data the firmware of both boards, containing the bootloader, was modified. The firmware is the first code to be executed upon device start. The focus of interest is the memory controller setup phase and as well as the DRAM initialization routine. During the DRAM initialization, the firmware itself does not require the use of DRAM or other storage. This makes it ideal for gathering PUF measurements in order to evaluate PUF characteristics.

In case of the Galileo platform, we modified the Quark EDKII firmware. At the end of the unmodified memory initialization, the DRAM self refresh is initialized (and ECC functionality is enabled if it is supported, like in Galileo). Prior to these two steps, code was implemented in order to measure the PUF comprising the following steps: writing initial value (*initval*) to the specific logical PUF (as defined by *addr* and *size*), waiting for time t, and then reading back the PUF response. After PUF response is processed, normal firmware execution and eventual boot up of the OS can resume. The implementation of firmware modification consists of about 60 lines of C code. Most of the code is responsible for setting PUF parameters and loops that write to and read from PUF. The PUF response is read back and printed out to the console for later analysis.

Likewise, for the PandaBoard we implemented a firmware-based solution that initializes the DRAM with the initialization value *initval*, disables auto-refresh of the memory controller, lets decay time t expire and finally sends the memory

contents over UART to our workstation. The implementation of firmware consists of about 50 lines (including writing, disabling refresh, waiting, and reading back memory).

Using this approach, however, the PUF can only be measured at boot up time and extends the boot up time by at least t seconds needed to decay the PUF. This is acceptable for testing, PUF metrics evaluation, and some scenarios. However, for practical and run-time use, we developed the kernel module.

3.2 Linux Kernel Module-based PUF Access

A kernel module to access logical PUFs is implemented for Galileo, which can be inserted at run time. The kernel module is designed with three phases. (1) Upon loading, the kernel module writes *initval* to memory address given by *addr* and *size*, and then modifies the memory controller via writes to configuration registers to disable automatic memory refresh (and disable ECC if needed). (2) The kernel module schedules [6] selective refresh function every Nms, where N is the desired refresh rate. Inside the selective refresh function a simple loop is used to loop over all memory addresses that need to be refreshed, issuing a read to memory word for every DRAM row. In Galileo, the DRAM modules used 1KB row buffers (8192 bits) [27]. (3) After decay time t seconds, the refresh cycles and ECC are enabled again and the PUF response is read out. Without the ballooning option the kernel module takes about 300 lines of C code in total. Ballooning code can be leveraged from existing virtualization projects, or a simpler kernel module can be used if `mem` option is used at boot-time to limit the size of system memory. The `mem` boot-time option effectively reservers a portion of memory as unusable by the system, ensuring that PUF, located in the unused memory, can be safely measured without any applications accessing that memory.

During PUF query, the OS and other applications are operating normally but some of CPU resources are spent on performing the selective memory refresh. Table 3 shows times required to do selective refresh on memory regions of various sizes. At room temperature, the 64ms refresh period is very conservative, and our experiments suggest that even with a refresh rate of 200ms our Linux setup runs stable. Previous work on DRAM retention time support our results [23]. Thus, depending on the operating conditions and required stability guarantees, the selective refresh period can be extended allowing for larger DRAM to be selectively refreshed or to selectively refresh DRAM less often. From Table 3 it can be seen that with active memory size over 128MB system will spend over 60% of time on in the selective refresh, leaving 40% of CPU time for other computations, when 64ms refresh target is selected. However, our experiments indicate that the OS and applications run stable also with smaller amount of memory, so reducing active memory to reduce selective refresh overhead is not a problem. We are even able to reduce the memory footprint of Yocto Linux, commonly

[6] A key feature of Linux, the so-called `workqueues` [53], allowing tasks to be scheduled at specific time intervals, is used.

Table 3: Time needed to perform memory reads to refresh varying sizes of memory regions on Intel Galileo board with DDR3 memory, last columns show CPU time spent in selective refresh assuming 64ms and 200ms refresh rate requirements.

Memory Size	**Selective Refresh Time**	**%CPU Time @ 64ms**	**%CPU Time @ 200ms**
32 MB	12.2 ms	19%	6%
64 MB	20.9 ms	32%	11%
128 MB	38.6 ms	60%	19%

Table 4: The evaluated logical PUF instances measured at $T = \{t_1 = 120s, t_2 = 180s, t_3 = 240s, t_4 = 300s, t_5 = 360s\}$. We denote a logical PUF as a single 32 KB segment on a given device.

Decay time t	Device	Num. Devices	Num. Logical PUFs Device	of Num. Measurements per	Maximum Fractional Intra-HD	Minimum Fractional Entropy
120s	PandaBoard	2	2	50	**0.0076**	**0.0071**
	IntelGalileo	2	2	50	**0.0004**	**0.0106**
180s	PandaBoard	2	2	50	**0.0222**	**0.0412**
	IntelGalileo	2	2	50	**0.0003**	**0.0250**
240s	PandaBoard	2	2	50	**0.0300**	**0.0685**
	IntelGalileo	2	2	50	**0.0010**	**0.0402**
300s	PandaBoard	2	2	50	**0.0361**	**0.0972**
	IntelGalileo	2	2	50	**0.0010**	**0.0594**
360s	PandaBoard	2	2	50	**0.0386**	**0.1216**
	IntelGalileo	2	2	50	**0.0015**	**0.0775**

used on Intel Quark devices, down to 32MB without any special modifications.[7] At 32MB, only 12.2ms are needed for the selective refresh, making more than 80% of processor's time available for other applications and almost 95% of time is available if the refresh period requirement is set at 200ms.

4 Evaluation of DRAM PUF Characteristics

We measured the PUF instances on Intel Galileo Gen2 and PandaBoard ES Rev. B3, as described before. We performed measurements on two different instances of each device type. Furthermore, given the large amount of memory present we measured two 32KB logical PUFs on each device. Each logical PUF was measured at five different decay times t with *initival* all 0s and all 1s. Based on these measurements we evaluated robustness, uniqueness and randomness, as well as time and temperature dependence and the stability. Table 4 summarizes the evaluated PUF instances.

Robustness – The intra-device Hamming distance (intra-HD) of multiple measurements obtained from a single PUF instance is used to quantize the inherent

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

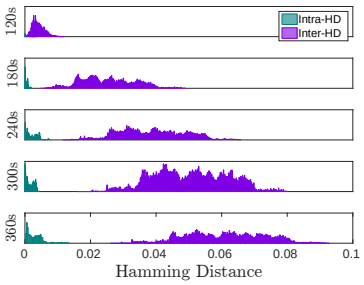

Fig. 3: Distribution of intra-HD and inter-HD values of the PandaBoards.

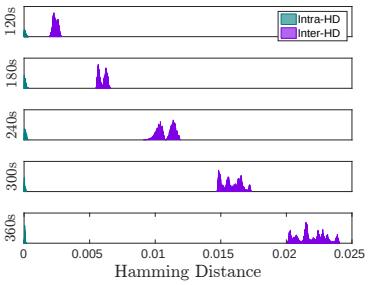

Fig. 4: Distribution of intra-HD and inter-HD values of the Intel Galileos.

noise (i.e., the robustness, and ideally should be close to zero). In order to calculate the fractional intra-HD, we listed all pairs of measurements from the same logical PUF. Next, we calculated the Hamming distance of each pair and divided the result by the length of measurements in bits in order to provide fractional metrics. The results are shown in Table 4.

Uniqueness – In order to evaluate the uniqueness of the PUF, the inter-device Hamming distance (inter-HD) is calculated on the basis of measurements from different PUF instances. The inter-HD was computed using the same approach as intra-HD. However, this time the calculations were performed on the basis of pairs of measurements obtained from different devices.

Figures 3 and 4 displays the distributions of intra-HD and inter-HD results for both device types. The intra-HD results suggest that responses from the same logical PUF have a limited amount of noise that can be easily corrected by standard Fuzzy Extractor schemes, thus allowing for robust identification. Furthermore, the inter-HD indicates responses from different PUFs exhibit differences that allow for uniquely identifying devices. A clear divide between the two distributions of hamming distances indicates that individual devices can well be distinguished, while same device is stable over many measurements.

Entropy – In order to generate cryptographic keys from the PUF response, the PUF measurements must exhibit sufficient entropy. We estimate Shannon entropy as follows. We consider a "pattern" as a PUF response after a decay time t with k cells that flipped. Assuming the flipped bits are distributed uniformly, given a logical PUF instance with a total of N cells and k flipped cells, the probability of observing one possible pattern p_i is:

$$P(p_i, t) = \frac{1}{\binom{N}{k}}. \tag{1}$$

Thus, the entropy of the PUF for a given decay time t can be calculated using:

$$H_t = log_2 \binom{N}{k}. \tag{2}$$

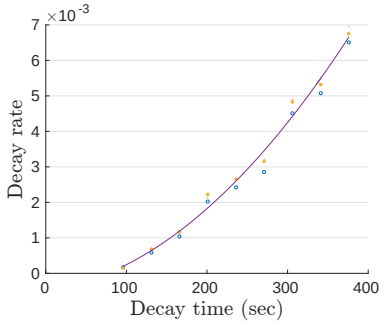

Fig. 5: Time-dependence of decay rate for DDR3 modules on Galileo at room temperature. Data points are fitted to a polynomial curve.

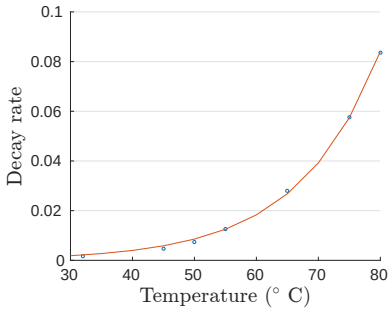

Fig. 6: Relation between the temperature of the DDR module and the decay rate measured at fixed time of 210s. Data points are fitted to an exponential curve.

Note, that simply observing the number of bit decaying at t is not sufficient for determining k, as the bit decay will be due to two effects: i) short-term noise that must be disregarded as well as ii) stable long-term decay characteristics that must be considered as k. In order to approximate k, multiple measurements for a single PUF can be averaged in order to eliminate the noise component. The results of the fractional entropy estimation are listed in Table 4.

Time and Temperature Dependent Decay – Figure 5 shows the decay rate as a function of decay time for the DDR3 memory modules from Micron on Intel Galileo. Measurements are taken at ambient room temperature with DRAM chips operating at around 32°C. Every data point shows the aggregated average of 8 measurements. Regarding the temperature effect on decay time ($e^{0.0625\Delta t}$), the result are similar as those of prior work (see [19], Fig. 2).

A second factor for the decay rate of DRAM cells is temperature. In Figure 6 we show the dependence between temperature and the decay rate for DDR3 modules on the Intel Galileo. In order to control the temperature, we used a metal ceramic heater to heat the surface of DRAM modules to defined temperature, and took all measurements with a decay time of 210s. Although the temperature affects the decay rate significantly, its effect is accelerating the decay. By using a equivalent decay time t' at temperature T', same PUF response can be got as with decay time t at temperature T. Hence, the temperature does not affect the inherent decay pattern.

Stability over Time – During extended lifetime of the devices, DRAM aging effects will begin to take place. Existing work on SRAM memory-based PUFs has explored aging effects [41, 26, 25, 35]. We are aware of limited work on aging-related effects in DRAM with regard to security [39]. Figure 7 shows the Hamming distances of measurements of same logical PUF taken 4 months apart on an Intel Galileo. Three logic PUFs were measured, and results are combined. Note that the measurements also include the noise introduced by temperature changes in our lab. The results suggest sufficient stability of DRAM PUFs over a long-term usage period.

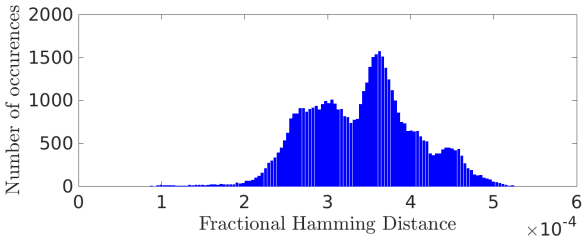

Fig. 7: Distribution of hamming distance of measurements taken from same PUF instances on Intel Galileo over four months.

Enrollment:

$\mathcal{SYS}$ creates/defines

$\mathcal{T} = \{t_{id,0}, t_{id,1}, \ldots, t_{id,n}\}$	...	set of decay times
$\mathcal{M} = \{m_{id,0}, m_{id,1}, \ldots, m_{id,n}\}$	...	set of PUF measurements
$\mathcal{W} = \{w_{id,0}, w_{id,1}, \ldots, w_{id,n}\}$	...	set of Helper Data
$\mathcal{K} = \{k_{id,0}, k_{id,1}, \ldots, k_{id,n}\}$	...	set of secret keys

$\mathcal{S}$ stores

$\mathcal{T}, \mathcal{M}, \mathcal{W}, H(\mathcal{K})$

$\mathcal{C}$ stores

$\mathcal{D}$

Fig. 8: Measurements generated and stored during PUF enrollment phase.

5 Lightweight Protocols for Device Authentication and Secure Channel Establishment

In the following Section we propose two protocols that use the idea of challenging a single DRAM-based PUF instance at different decay times $t_x, t_{x+1}, \ldots, t_n$. 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-based PUF instance 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 being $[addr, size, initval]$ (see Section 2.1), we refer to it using the abbreviated id.

Enrollment: An enrollment phase precedes both protocols, which is assumed to be conducted at a trusted party $\mathcal{SYS}$ that corresponds to a manufacturer of $\mathcal{D}$ or a system integrator. The authentication protocol is based on a security parameter ϵ_{bits} which can be changed to adjust security and usability of the protocol. The selection of a reasonable value of the parameter is discussed at the end of this section.

During the enrollment phase, a trusted party $\mathcal{SYS}$ queries n times the PUF instance with ID id found on device $\mathcal{D}$ in order to get a set of measurements $\mathcal{M} = \{m_{id,0}, m_{id,1}, ..., m_{id,n}\}$ at a defined set of decay time $\mathcal{T} = \{t_{id,0}, t_{id,1}, ..., t_{id,n}\}$, i.e.: $m_{id,x} = PUF(t_{id,x})$. Here, $t_{id,0}, t_{id,1}, ..., t_{id,n}$ are cho-

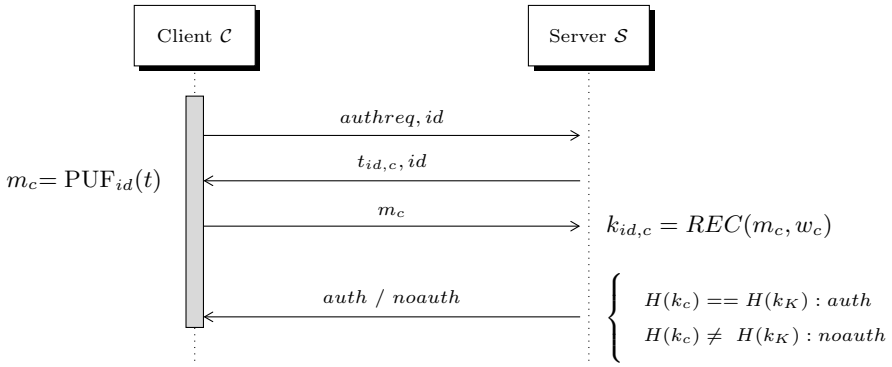

Fig. 9: Sequence diagram of the device authentication protocol.

sen, such that $t_{id,0} < t_{id,1} < ... < t_{id,n}$ and for every tuple of subsequent decay times the number of new bit flips is always greater then a given security parameter *secpone*: $(t_x, t_{x+1}) \geq \epsilon_{bits}$. [8] Finally, $\mathcal{SYS}$ uses $\mathcal{M}$ as input to the $GEN(\cdot)$ function of a Fuzzy Extractor [6] to create a set of Helper Data $\mathcal{W}$ and a set $\mathcal{K}$ containing uniformly distributed keys, that correspond to $\mathcal{M}$: $\mathcal{W} = \{w_{id,0}, w_{id,1}, ..., w_{id,n}\}$ and $K = \{k_{id,0}, k_{id,1}, ..., k_{id,n}\}$, such that $(k_{id,x}, w_{id,x}) = GEN(m_{id,x})$. Further, the hashes $H(\mathcal{K})$ of $\mathcal{K}$ are generated. Eventually, $\mathcal{T}$, $\mathcal{M}$, $\mathcal{W}$ and $H(\mathcal{K})$ will be given to $\mathcal{S}$ whilst $\mathcal{D}$ will be handed to $\mathcal{C}$ in a secure manner. The measurements are summarized in Figure 8.

Device Authentication: In order to authenticate the client $\mathcal{C}$ towards an honest server $\mathcal{S}$ the server randomly chooses an unused $t_{id,c}$ that is greater than all decay times used previously for that PUF. Next, $\mathcal{S}$ transmits $t_{id,c}$ to $\mathcal{C}$, who uses it as input to his PUF to retrieve a measurement m_c, which is sent back to $\mathcal{S}$. The server in turn uses the Fuzzy Extractor with m_c as input to reconstruct a key based on the PUF measurement received from $\mathcal{C}$: $k_{id,c} = REC(m_c, w_c)$. Lastly, $\mathcal{S}$ compares the hash of the reconstructed key with the one stored in $H(\mathcal{K})$. If both match, $\mathcal{C}$ is authenticated, otherwise the client is not authenticated. The authentication protocol is depicted in Figure 9.

The authentication is extremely lightweight in terms of computational overhead, memory footprint and round-trip. 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: For other use cases, device authentication may not suffice and instead a secure communication channel is required. In this case, the protocol depicted in Figure 10 establishes a shared key between $\mathcal{C}$ and $\mathcal{S}$. The basic idea is to apply a Fuzzy Extractor on measurements in order to

[8] If $t_{x1} \leq t_{x2}$ and $addr_{x1} = addr_{x2}, size_{x1} = size_{x2}, initval_{x1} = initval_{x2}$, we observe $m_{x1} \subseteq m_{x2}$. So security is provided by new flipped bits between t_x and t_{x+1}.

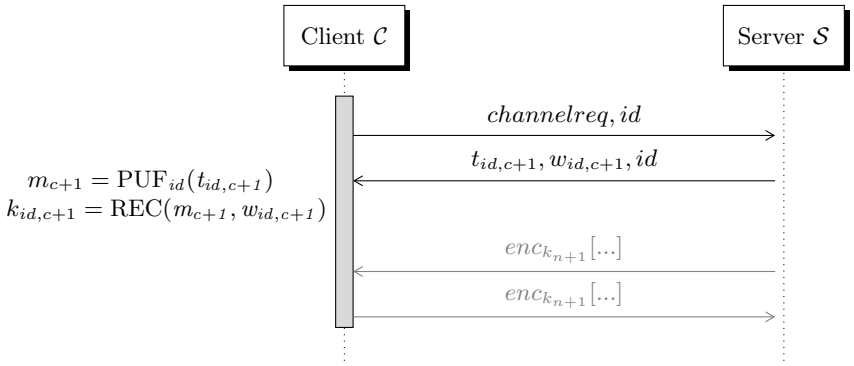

Fig. 10: Sequence diagram of the secure channel establishment protocol. If in the course of communication between client and server new session keys must be negotiated, the server can send unused decay times t_{c+j}, corresponding Helper Data w_{c+j} as well as id information to identify the PUF.

create a shared key at the client using Helper Data sent on-demand. In particular, the server sends a fresh decay time that again is greater then all previously used decay times $t_{id,c+1}$ along with the corresponding Helper Data $w_{id,c+1}$. The client evaluates his PUF instance using $t_{id,c+1}$ in order to retrieve m_{c+1}, which is used in combination with $w_{id,c+1}$ to reconstruct k_{c+1}: $k_{c+1} = REC(m_{c+1}, w_{id,c+1})$. The server executes the same process using $t_{id,c+1}$ and the corresponding enrolled measurement that was stored in $\mathcal{M}$.

On The Choice of Security Parameters: The protocols involve a security parameter (ϵ_{bits}) that allows for a trade-off between security and usability. Primarily, ϵ_{bits} depicts the number of *new* bits that flipped between two successive decay times t_x, t_{x+1}. In other words, the set of measurements M must be created such that in every new measurement $m_x + 1$ at least ϵ_{bits} new bits have flipped compared to measurement m_x. As the attacker knows m_x the security of the protocol ultimately is based on the number of guesses an adversary has to conduct, to infer the new locations of bit flips in m_{x+1} from m_x. Given that the space of potential *new* bit flips is of size N, the attacker would have to search the whole space N for all possible combinations of l new bit flips. Providing 128-bit security, the resulting search space will be approximately equal to[9]:

$$\binom{N}{l} \geq 2^{128}. \tag{3}$$

Given the size of a logical PUF of $N = 32$ KiB (262144 bits) and a required security level of 128 bit the equation is satisfied with $l \geq 8$:

$$\binom{262144}{8} \geq 2^{128}. \tag{4}$$

[9] In fact, N is decreasing with growing x for $(t_x, t_{x+1}, ..., t_n)$ (i.e. as time passes by). However, given the huge total size of available bits and the comparatively small number of bit flips, the behavior approximates to Equation 3.

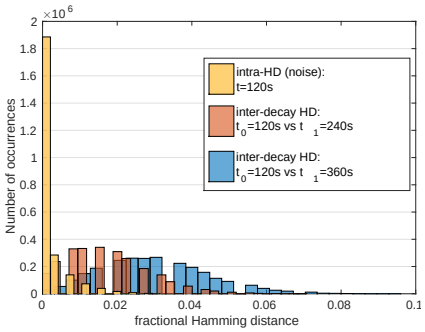

Fig. 11: Histograms of: the error due to inherent noise, inter-HD of measurements with decay rates 120s vs. 240s, and of measurements with decay rates 120s vs. 360s at block level with block size = 247bits.

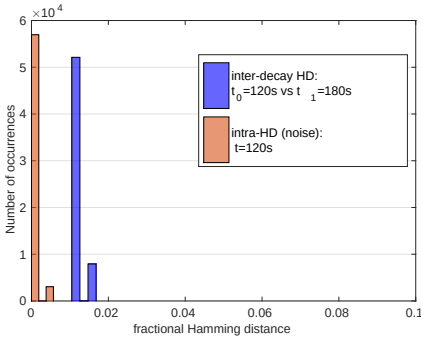

Fig. 12: Distributions of intra-HD and the inter-decay-HD at block level with block size = 247bits.

Adversary and Threat Model: Our adversary model for the protocols considers two types of adversaries: i) a passive network eavesdropper $\mathcal{A}_E$ and ii) an active network adversary $\mathcal{A}_A$.

The weaker adversary $\mathcal{A}_E$ is only able to passively observe the network traffic between client and server, capturing transmitted messages over the course of the protocol run. $\mathcal{A}_E$ is not able to access the DRAM of the client device and thus cannot query the PUF. For messages that are sent over a secured channel, we assume that both parties employ strong symmetric crypto primitives, providing at least 128-bit security using the key derived from the PUF instance. Thus, unless the adversary is able to observe plaintext messages, or is able to get the symmetric keys by means that are out of scope of this protocol, he or she is not able to decrypt any messages.

If the $\mathcal{A}_E$ attacker captures a protocol transcript of (multiple) authentication trials and an additional run of the secure establishment protocol, he is in possession of several recently used PUF measurements at $T_{old} = t_{id,c-k}, \ldots, t_{id,c-1}$ and Helper Data $w_{id,c}$ that corresponds to a successive measurement $m_{id,c}$. Whilst the current PUF measurement is not public, $\mathcal{A}_E$ could try to reconstruct bits of keys $k_{id,c}$ using the captured older measurements, e.g. $m_{id,c-1}$, in combination with the current Helper Data $w_{id,c}$. At first glance, this method seems to be successful as subsequent measurements share the some of the bits ($m_{id,c-1} \subseteq m_{id,c}$) and only a small fraction of new bits are introduced in a new measurement. Especially, in some cases the Hamming distance between measurements taken at different decay times $t_{c-n}, \ldots, t_{c-1}$ may be smaller or at least equal to the inherent noise of any measurement, which needs to be corrected during the ECC decoding step of the Fuzzy Extractor (FE). Thus, using $w_{id,c}$ and a previous measurement $m_{id,c-1}$, the ECC would map the measurement to the same error code domain, thus leaking bits of $k_{id,c-1}$. Typically, linear block-codes operate

on message blocks that are much smaller compared to the 32KB logic PUF,[10] and thus, only a few bits in a block are flipped, which enlarge the effect of noise. We computed the distribution of the intra-HD (i.e. the noise) against the Hamming distances of blocks of measurements obtained from several pairs of decay times. The results for the Galileo board and a message block size of 247 that correspond to the BCH(255,247,1) code are shown in Figure 11. They show an overlapping area of both distributions and thus confirm the flaw. In order to mitigate this problem, only such blocks that maximize the distance between the noise error and the Hamming distance of measurements from different decay times can be selected as input to the Fuzzy Extractor. Such blocks can be identified during the enrollment phase and data structures that store pointers to the respective blocks can be added to the corresponding Helper Data w_c. Using the block selection approach, we indeed achieve that the newly emerged bit flips of measurements of subsequent decay times cannot be used in combination with previous Helper Data in order to leak bits of keys of previous measurements. The results are shown in Figure 12.

The active adversary $\mathcal{A}_A$ is more powerful and is able to inject, replay or suppress messages. In the case that $\mathcal{A}_A$ pretends to be a server, he or she could try to send challenges to the device and thus trick the client to provide PUF responses. However, it is device that initiates the authentication. Thus device can limit how often it communicates with server, extensively limiting the rate at which PUF can be probed. If $\mathcal{A}_A$ pretends to be a client, he or she could try to send messages to the server and use the server as an oracle and gain information. However, the search space is $\binom{N}{l}$, which is big enough as discussed earlier in this section. Also, for secure channel, by design the helper data does not leak key bits, so the adversary still have to brute force the PUF response.

6 Related Work

A Physically Unclonable Function (PUF) is a unique and stable physical characteristic of a piece of hardware, due to variations in the fabrication processes. Different PUF implementations have been proposed and PUFs can be categorized into delay-based [10] as well as memory-based [38] PUFs.

Delay-based PUFs use minuscule differences in the delays a signal exhibits to traverse a set of identically laid out circuits to extract a PUF response. They often require dedicated circuits, such as arbiters and ring oscillators [42]. In contrast, memory-based intrinsic PUFs leverage variations in storage cells of memory elements already present on the computing devices, most common types are based on SRAM, and more recently on DRAM as in our work.

Given their favorable characteristic of unclonability, PUFs have been proposed as lightweight cryptographic building blocks used in security primitives and protocols. Subsequently, PUFs have been used for lightweight authentication and identification [20, 46], hardware-software binding [14, 34, 21, 49, 36], remote

[10] BCH codes usually have a maximum block size of 1013 bits [44].

attestation [22, 40]. PUFs have also been used for secret key storage, also referred to as a physically obfuscated key (POK) [48, 47], and various other applications. The applicability of PUFs to such security-related scenarios highly depends on the adversary's ability to phenomenologically clone the PUF, primarily by emulating the entirety of the PUF's challenge-response space.

6.1 DRAM-based PUFs

The earliest approach to exploit manufacturing variations of DRAM cells for identification and random number generation was proposed in [30] where the authors create DRAM fingerprints to mitigate hardware counterfeiting. In subsequent work, [18] introduced the concept of using the DRAM cells' charge decay for extracting random bits as well as unique identifiers from external DDR3 modules. An FPGA-based setup was used that would query an external DRAM module to overcome the hurdles introduced by commercial CPUs and SoC while disabling the refresh rate. In [24] a secure key storage scheme based on DRAM modules was suggested. Again, to implement the functionality of disabling and enabling the DRAM's supply power, the authors used an FPGA-based setup combined with external DDR3 SODIMM modules. In [43] the start-up values of the DRAM cell were shown to have similar behavior to SRAM-based PUF as DRAM cells have a unique tendency to initialize to either to zero or one, thus allowing for extracting a PUF instance. The authors prove the existence of a DRAM PUF based on start-up values by evaluating a set of external DRAM modules read out by an FPGA. In [15] an authentication scheme based on signatures generated by DRAM PUFs is presented. The authors introduce PUF-like characteristics to DRAM arrays by shortening the write duty-cycle, thus inducing failed write operations that are bound to process variations. The experiments are simulated using a circuit simulator and shows robust as well as unique PUF responses that have high entropy. Rehmati et. al [29] make use of the error patten in approximate DRAM as system identifying fingerprint. To our knowledge, no prior work has shown a DRAM PUF in commodity off-the-shelf devices that did not require custom FPGA setup. Furthermore, no run-time accessible DRAM PUF has been presented before, as we do.

7 Conclusion

In this work we presented intrinsic PUFs that can be extracted from dynamic random-access memory (DRAM) in commodity devices, and accessed at runtime via a Linux kernel module. An evaluation of the DRAM PUFs on unmodified, commodity devices such as PandaBoard and Intel Gallileo 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 OS kernel module. The construction of a practical DRAM

PUF that can be queried at runtime was based on ideas of *selective DRAM refresh* and *memory ballooning*. Based on the DRAM PUFs, design of protocols that leverage the PUF to enable device authentication and identification were presented and evaluated. As an intrinsic PUF, DRAM PUFs overcome limitations of SRAM, especially with regard to expanded challenge-response space and ability to be accessed at run time. Consequently, our work presents a new alternative for device authentication and secure communication by leveraging DRAM already present in commodity devices, while enabling normal Linux OS operation when portions of the DRAM are utilized as a PUF.

8 acknowledgements

The authors would like to thank Intel for generously donating the Galileo Gen2 boards, and Kevin Ryan and Ethan Weinberger for their help on building the heating circuit.

This work has been co-funded by the DFG as part of project P3 within the CRC 1119 CROSSING.

References

1. Google nest learning thermostat, https://nest.com/thermostat/meet-nest-thermostat/, accessed Feb. 2016
2. Hacking defcon 23's iot village samsung fridge, https://www.pentestpartners.com/blog/hacking-defcon-23s-iot-village-samsung-fridge/, accessed Feb. 2016
3. 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)
4. Bacha, A., Teodorescu, R.: Authenticache: harnessing cache ecc for system authentication. In: Proceedings of the 48th International Symposium on Microarchitecture. pp. 128–140. ACM (2015)
5. 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 4(2), 77–89 (2014)
6. Dodis, Y., Reyzin, L., Smith, A.: Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In: Advances in cryptology-Eurocrypt 2004. pp. 523–540. Springer (2004)
7. Foster, I., Prudhomme, A., Koscher, K., Savage, S.: Fast and vulnerable: a story of telematic failures. In: 9th USENIX Workshop on Offensive Technologies (WOOT 15) (2015)
8. Intel® galileo gen 2 development board, http://www.intel.de/content/www/de/de/embedded/products/galileo/galileo-overview.html accessed Feb. 2016
9. Pandaboard, http://www.pandaboard.org accessed Feb. 2016
10. Gassend, B., Clarke, D., Van Dijk, M., Devadas, S.: Delay-based circuit authentication and applications. In: Proceedings of the 2003 ACM symposium on Applied computing. pp. 294–301. ACM (2003)
11. Golz, J., Safran, J., He, B., Leu, D., Yin, M., Weaver, T., Vehabovic, A., Sun, Y., Cestero, A., Himmel, B., et al.: 3d stackable 32nm high-k/metal gate soi embedded dram prototype. In: 2011 Symposium on VLSI Circuits-Digest of Technical Papers (2011)
12. Greenberg, A.: Hackers remotely kill a jeep on the highwaywith me in it. Wired, 21July (2015)
13. Guajardo, J., Kumar, S.S., Schrijen, G.J., Tuyls, P.: FPGA intrinsic PUFs and their use for IP protection. Springer (2007)
14. Guajardo, J., Kumar, S.S., Schrijen, G.J., Tuyls, P.: Brand and ip protection with physical unclonable functions. In: Circuits and Systems, 2008. ISCAS 2008. IEEE International Symposium on. pp. 3186–3189. IEEE (2008)
15. 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 2015 Design, Automation & Test in Europe Conference & Exhibition. pp. 647–652. EDA Consortium (2015)
16. Hernandez, G., Arias, O., Buentello, D., Jin, Y.: Smart nest thermostat: A smart spy in your home. Black Hat USA (2014)
17. Keeth, B.: DRAM circuit design: fundamental and high-speed topics, vol. 13. John Wiley & Sons (2008)

18. 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: Circuits and Systems (ISCAS), 2014 IEEE International Symposium on. pp. 2740–2743. IEEE (2014)
19. Kim, K., Lee, J.: A new investigation of data retention time in truly nanoscaled drams. Electron Device Letters, IEEE 30(8), 846–848 (2009)
20. Kocabaş, Ü., Peter, A., Katzenbeisser, S., Sadeghi, A.R.: Converse PUF-based authentication. Springer (2012)
21. 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)
22. Kong, J., Koushanfar, F., Pendyala, P.K., Sadeghi, A.R., Wachsmann, C.: Pufatt: Embedded platform attestation based on novel processor-based pufs. In: Design Automation Conference (DAC), 2014 51st ACM/EDAC/IEEE. pp. 1–6. IEEE (2014)
23. 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. vol. 41, pp. 60–71. ACM (2013)
24. Liu, W., Zhang, Z., Li, M., Liu, Z.: A trustworthy key generation prototype based on ddr3 puf for wireless sensor networks. Sensors 14(7), 11542–11556 (2014)
25. Maes, R., van der Leest, V.: Countering the effects of silicon aging on sram pufs. In: Hardware-Oriented Security and Trust (HOST), 2014 IEEE International Symposium on. pp. 148–153. IEEE (2014)
26. 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: ESSCIRC (ESSCIRC), 2012 Proceedings of the. pp. 486–489. IEEE (2012)
27. Micron ddr3l datasheet, `https://www.micron.com/~/media/documents/products/data-sheet/dram/ddr3/4gb_1_35v_ddr3l.pdf`, accessed Nov. 2015
28. Phone as a token - turn your phone into an authentication token, `https://www.intrinsic-id.com/technology/phone-as-a-token/`, accessed Feb. 2016
29. Rahmati, A., Hicks, M., Holcomb, D.E., Fu, K.: Probable cause: the deanonymizing effects of approximate dram. In: Proceedings of the 42nd Annual International Symposium on Computer Architecture. pp. 604–615. ACM (2015)
30. 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. Solid-State Circuits, IEEE Journal of 48(11), 2934–2943 (2013)
31. 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. Solid-State Circuits, IEEE Journal of 48(4), 940–947 (2013)
32. Rührmair, U., Busch, H., Katzenbeisser, S.: Strong pufs: models, constructions, and security proofs. In: Towards hardware-intrinsic security, pp. 79–96. Springer (2010)
33. Ruhrmair, U., Holcomb, D.E.: Pufs at a glance. In: Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014. pp. 1–6. IEEE (2014)
34. 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)
35. Schaller, A., Škorić, B., Katzenbeisser, S.: On the systematic drift of physically unclonable functions due to aging. In: Proceedings of the 5th International Workshop on Trustworthy Embedded Devices. pp. 15–20. ACM (2015)

36. Scheel, R.A., Tyagi, A.: Characterizing composite user-device touchscreen physical unclonable functions (pufs) for mobile device authentication. In: Proceedings of the 5th International Workshop on Trustworthy Embedded Devices. pp. 3–13. ACM (2015)
37. Schneier, B.: The internet of things is wildly insecure—and often unpatchable. Wired, Jan (2014)
38. 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)
39. Schroeder, B., Pinheiro, E., Weber, W.D.: Dram errors in the wild: a large-scale field study. In: ACM SIGMETRICS Performance Evaluation Review. vol. 37, pp. 193–204. ACM (2009)
40. Schulz, S., Sadeghi, A.R., Wachsmann, C.: Short paper: lightweight remote attestation using physical functions. In: Proceedings of the fourth ACM conference on Wireless network security. pp. 109–114. ACM (2011)
41. 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: Circuits and Systems (ISCAS), 2011 IEEE International Symposium on. pp. 567–570. IEEE (2011)
42. Suh, G.E., Devadas, S.: Physical unclonable functions for device authentication and secret key generation. In: Proceedings of the 44th annual Design Automation Conference. pp. 9–14. ACM (2007)
43. Tehranipoor, F., Karimina, N., Xiao, K., Chandy, J.: Dram based intrinsic physical unclonable functions for system level security. In: Proceedings of the 25th edition on Great Lakes Symposium on VLSI. pp. 15–20. ACM (2015)
44. Thamer: Bch codes. University Lecture, `https://nest.com/thermostat/meet-nest-thermostat/`, accessed March 2016
45. 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 March 2016
46. Tuyls, P., Batina, L.: Rfid-tags for anti-counterfeiting. In: Topics in cryptology–CT-RSA 2006, pp. 115–131. Springer (2006)
47. 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)
48. Tuyls, P., Škorić, B.: Secret key generation from classical physics: Physical uncloneable functions. In: AmIware Hardware Technology Drivers of Ambient Intelligence, pp. 421–447. Springer (2006)
49. Van Aubel, P., Bernstein, D.J., Niederhagen, R.: Investigating sram pufs in large cpus and gpus. In: Security, Privacy, and Applied Cryptography Engineering, pp. 228–247. Springer (2015)
50. Viega, J., Thompson, H.: The State of Embedded-Device Security (Spoiler Alert: It's Bad). IEEE Security & Privacy (5), 68–70 (2012)
51. Waldspurger, C.A.: Memory resource management in vmware esx server. SIGOPS Oper. Syst. Rev. 36(SI), 181–194 (Dec 2002)
52. Wordeman, M., Silberman, J., Maier, G., Scheuermann, M.: A 3d system prototype of an edram cache stacked over processor-like logic using through-silicon vias. In: Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2012 IEEE International. pp. 186–187. IEEE (2012)

53. Linux workqueues, http://www.makelinux.net/ldd3/chp-7-sect-6, accessed Nov. 2015