# On the Effectiveness of the Remanence Decay Side-Channel to Clone Memory-based PUFs

Yossef Oren[1], Ahmad-Reza Sadeghi[2], and Christian Wachsmann[3]

[1] Tel-Aviv University, Israel
yos@eng.tau.ac.il

[2] TU Darmstadt/CASED, Germany
ahmad.sadeghi@trust.cased.de

[3] Intel CRI-SC at TU Darmstadt, Germany
christian.wachsmann@trust.cased.de

**Abstract.** We present a side-channel attack based on remanence decay in volatile memory and show how it can be exploited effectively to launch a non-invasive cloning attack against SRAM PUFs — an important class of PUFs typically proposed as lightweight security primitive with low overhead by using the existing memory of the underlying device. We validate our approach against two SRAM PUF implementations in 65 nm CMOS ASICs. We discuss countermeasures against our attack and propose the constructive use of remanence decay to improve the cloning-resistance of SRAM PUFs.
Moreover, as a further contribution of independent interest, we show how to use our evaluation results to significantly improve the performance of the recently proposed TARDIS scheme, which is based on remanence decay in SRAM and used as a time-keeping mechanism for low-power clock-less devices.

**Keywords:** SRAM PUF, fault injection attack, side-channel analysis, data remanence decay

## 1 Introduction

Physically Unclonable Functions (PUFs) have become an attractive research area and are increasingly proposed as building blocks in cryptographic protocols and security architectures. One major class of PUFs and the focus of this paper are memory-based PUFs [6,20,28,17,10,18]. These PUFs are commonly proposed as an alternative to secure non-volatile memory and are used in a variety of anti-counterfeiting mechanisms and authentication schemes [19,30,6,7,24,25,4].

Memory-based PUFs are arrays of volatile memory elements, such as SRAM cells [6,10], flip-flops [20,18] or latches [28,17]. These elements typically are bi-stable circuits with two stable states corresponding to a logical 0 and 1. By applying an external control voltage to the inputs of the element, it can be forced to enter either of the two states. Memory-based PUFs exploit the following phenomena: When powering up such an element without applying an external

control voltage, its state mainly depends on the physical characteristics of the underlying transistors. Due to uncontrollable manufacturing variations, these characteristics are unique for each physical instantiation of the element. Hence, the state of all memory elements (after powering the memory without applying a control voltage) can be used as a unique identifier (called the PUF response) for the device containing the memory. However, since the PUF response could be read out completely and copied to another device, a fundamental requirement on the implementation of memory-based PUFs is to prevent unintended/unauthorized access to the PUF response. This requirement is indeed debatable, since it implies the underlying memory to be tamper-evident and the presence of some security mechanism protecting the PUF response against unintended access.

Memory-based PUFs are considered as very cost-effective by using the existing memory of the device they are integrated in [30,6,25,5,13,16,15]. However, in this case the memory is also used to store the data of some other component in the device and will at some point be overwritten with the data of this component. In particular, volatile memory is typically initialized, i.e., overwritten with a *known* bit pattern (usually all zeros or ones), before it is used as a data storage. Further, although volatile memory loses the data it stores when it is powered off, the data is not immediately lost but decays slowly over time [8,23]. Hence, it is very likely that any data written to the memory of a memory-based PUF may affect the PUF's response when the power has been removed only for a short amount of time. Although this effect has been discussed in the literature [29,26,10,27,11], it has never been used to attack memory-based PUFs.

**Contribution.** We present the first fault injection attack based on remanence decay in volatile memory, and show how it can be exploited for a non-invasive cloning attack against SRAM-based PUFs. To the best of our knowledge this is the first cloning attack on memory-based PUFs based on remanence decay. In particular, our contribution is as follows:

*First cloning attack on SRAM PUFs using remanence decay side channels.* Our attack recovers the secret response of a memory-based PUF in applications where the underlying memory is overwritten with a known value after the PUF response has been read. This attack can be applied but is not limited to all memory-PUF based systems that share the PUF memory with some other functionality, which is often suggested [30,6,25,5,13,16,15] to allow for cost-effective PUF implementations. We show that the attack is successful against small memory-based PUFs even when using common lab equipment. The only requirements of the attack are that the adversary can control the supply voltage of the device containing the PUF and that the PUF memory is initialized with a known value before it is used as a data storage, which typically is the case.

*Experimental validation of the attack.* We validated the feasibility of our attack using two SRAM PUF implementations in two 65 nm ASICs and suggest several improvements to the test setup to increase the performance of our attack.

*Constructive use of remanence decay.* We propose using remanence decay as a source of side-channel information to enhance the cloning-resistance of SRAM PUFs. Cloning such a PUF would require emulating the remanence decay behavior, which increases the costs of a clone and may render cloning uneconomical.

*Improved TARDIS time-keeping mechanism.* As a contribution of independent interest, we propose a time-memory tradeoff to dramatically reduce the complexity of the recently proposed TARDIS [23] time-keeping mechanism for clock-less devices from linear to logarithmic time, enhancing its applicability to many practical scenarios.

**Outline.** We introduce our notation and the system and adversary model in Section 2. The attack is described in Section 3 and its experimental validation is presented in Section 4. A practical instantiation of our attack is shown in Section 5. We discuss the impact and improvements of the attack in Section 6 and make suggestions on the constructive use of remanence decay, including the improved TARDIS algorithm, in Section 7. We give an overview of the related work in Section 8 and finally conclude in Section 9.

## 2 Model and Preliminaries

We consider devices that contain a memory-based PUF and overwrite the underlying memory with a known value after the PUF response has been read. This typically happens when the PUF memory is also used as a data storage for some other functionality in the device, which is a common approach [30,6,25,5,13,16,15] to cost-effective implementations of memory-based PUFs.

**Initial State.** Volatile memory is typically initialized, i.e., overwritten with a specific bit pattern (usually all zeroes or ones), before it is used as a data storage. We denote this pattern as the *initial state* of the memory.

**Definition 1 (Initial State).** *The* initial state *of the memory is the matrix* $\boldsymbol{M}_{\text{Init}}$ *representing the data that is written to the memory before it is used as a data storage, i.e., after the memory has been used as a PUF.*

**Start-up State.** Observe that the data stored in volatile memory is typically not immediately lost when the power to the memory is removed but decays slowly over time [8,23]. Hence, when powered off only for a short time, the memory may still hold some of the data that has been written to it before the power-cycle. We capture this aspect by introducing the notion of the *start-up state*.

**Definition 2 (Start-up State).** *Let* $v_{\text{nom}}$ *be the nominal supply voltage of the memory. Consider the following experiment:*

1. *Set the supply voltage of the memory to 0 V for time* $t$

2. *Set the supply voltage of the memory to* $v_{\text{nom}}$
3. *Read the states of all memory elements and store them in a matrix* $\boldsymbol{M}_t$

*We say that* $\boldsymbol{M}_t$ *is the* start-up state *of the memory with respect to the time* $t$*.*

**PUF State.** The response of a memory-based PUF corresponds to the start-up state of the underlying memory, where the memory has been powered off long enough that any data previously stored in it has decayed. We capture this aspect by introducing the notion of the *PUF state* of a memory.

**Definition 3 (PUF State).** *Let* $t_\infty$ *be the time the memory must be without power for any data previously stored in it to be decayed. We denote the start-up state* $\boldsymbol{M}_{t_\infty}$ *as the* PUF state $\boldsymbol{M}_{\text{PUF}}$ *of the memory, i.e.,* $\boldsymbol{M}_{\text{PUF}} := \boldsymbol{M}_{t_\infty}$*.*

Observe that, in case the memory has been powered off only for a short time before it is used as a PUF, the PUF response may be distorted by the data previously stored in the memory.

**Device Behavior.** At some point while the device is running, it reads the start-up state of its memory and uses it as the PUF response in some computation. In many applications the result of this computation can be observed from outside the device. For instance, in PUF-based (authentication) protocols [30,25,4], the device receives some query $Q$ and responds with a message $X$ that depends on the PUF response. In these schemes, the response of the memory-based PUF is typically used to derive a cryptographic secret that is used to compute $X$. However, the device behavior is not limited to challenge-response protocols. In the extreme case $X$ could be only one single bit of information, e.g., indicating whether the correct PUF response was extracted from the memory or not. For instance, in PUF-based IP protection schemes [6,7,24], the device refuses to boot in case the PUF response is incorrect, which can be observed by the adversary. We capture this aspect by introducing the notion of *device behavior*.

**Definition 4 (Device Behavior).** *Let* $\boldsymbol{M}_t$ *be the start-up state (Definition 2) of the device memory with respect to some time* $t$*. Further, let* $Q$ *be some query that can be sent to the device. We denote with* $X = \mathsf{Dev}(\boldsymbol{M}_t, Q)$ *the response to* $Q$ *of the device using the start-up state* $\boldsymbol{M}_t$*. The algorithm* $\mathsf{Dev}$ *describes the* behavior *of the device with respect to* $Q$ *and* $\boldsymbol{M}_t$*.*

**Assumptions and Adversary Model.** Following the common adversary model of memory-PUFs [30,6,25,5,13,16,15], we assume that the adversary cannot simply read the plain PUF response from the underlying memory. This means that the adversary does *not* know the start-up state $\boldsymbol{M}_t$ (Definition 2) with respect to any time $t$ and, in particular, he does not know the PUF state $\boldsymbol{M}_{\text{PUF}}$ (Definition 3). Further, we assume that all algorithms implemented in the device are known to the adversary (Kerckhoffs' principle). This means that the adversary could compute $X = \mathsf{Dev}(\boldsymbol{M}_t, Q)$ if he knew $\boldsymbol{M}_t$ and $Q$. Moreover, the adversary

knows the initial state $\boldsymbol{M}_{\text{Init}}$ (Definition 1) that is part of the algorithms used by the device. Furthermore, we assume that the adversary can observe the device behavior (Definition 4) and that he can control the time $t$ the memory is powered off before it is used as a PUF. That is, the adversary can send some query $Q$ to the device and observe its reaction/answer $X$ that depends on the device's start-up state $\boldsymbol{M}_t$.

## 3 Cloning SRAM PUFs Using Remanence Decay

The high level idea and approach of our attack is to recover the PUF response in a device that overwrites the SRAM of the PUF with some data that is known to the adversary (cf. Section 2). The attack principle is similar to the attack by Biham and Shamir [2] (which we call Biham-Shamir attack in the following) to extract a secret key stored in some device (e.g., a smart card).

The Biham-Shamir attack consists of two phases: In the first phase, the adversary collects a sequence of ciphertexts, each encrypting the same plaintext with a slightly different key. More detailed, the adversary requests the device to encrypt the plaintext and, after he receives the corresponding ciphertext, he injects a fault into the device that sets one bit of the key to a known value. The adversary repeats this step until he set all the bits in the key to a known value. In the second phase of the attack, the adversary iteratively recovers the secret key of the device. More detailed, starting from the ciphertext that has been generated by the device using the known key, the adversary performs an exhaustive search for the key used by the device to generate each ciphertext collected in the first phase. Since the keys of two consecutive ciphertexts differ in at most one single bit and the value of this bit is known to the adversary, this exhaustive search is linear in the bit-length of the key. This way, the adversary can recover the secret key of the device with a total effort quadratic in the bit-length of the key.

Similarly we aim at extracting the secret PUF state from a device containing an SRAM PUF. Similar to the Biham-Shamir attack, we iteratively collect a series of device responses to the same query, each generated using a different start-up state. In each iteration, we send the query to the device, record its response (that depends on the start-up state), and then inject a fault to change some bits in the start-up state. The fault injection is performed by carefully controlling the amount of remanence decay undergone by the SRAM, e.g., by increasing the time the SRAM is powered off between two iterations. This has the effect that, due to the different decay times of the SRAM cells, some cells lose the known value of the initial state and revert back to their unknown PUF state, while others still keep their initial state. Further, the cells do not immediately revert to their PUF state but there is a short transition phase where the memory cell is *metastable* and takes a random state. Hence, in contrast to the Biham-Shamir attack, the number of bits $k$ that are different in the start-up states used in two consecutive iterations is typically larger than one bit. However, as we show in Section 4, $k$ has an upper bound that highly depends on the method and the accuracy of the equipment used to control the remanence decay.

In the second phase of the attack, we iteratively recover the unknown PUF state starting from the known initial state. A trivial approach would be to perform a simple exhaustive search for all cells that have reverted to their PUF state in the start-up states of two consecutive iterations of phase one. However, while this approach works for small values of $k$, it is inefficient for large values of $k$. In Section 6.2, we discuss several approaches to reduce the value of $k$ by improving the test setup and to reduce the complexity of the search for the changed bit positions. Before we describe our attack in detail, we first explain the underlying requirements and building blocks.

### 3.1 Controlling the Remanence Decay

An essential requirement for our attack is that the adversary can precisely control the remanence decay in the SRAM. There are two approaches how this can be achieved. The *voltage-based* approach directly changes the supply voltage to the chip for a certain amount of time, while the *time-based* approach sets the supply voltage of the chip to 0 V for a precisely-measured amount of time. In general, the time-based approach is easier to use since it only requires a precise timer to trigger the voltage drop, while the voltage-based approach requires an expensive and precise digital-to-analog converter. For this reason, we focus on the time-based approach.

### 3.2 Data Remanence Experiment

One major building block of our attack is the data remanence experiment where the adversary observes how the remanence decay affects the behavior of the device containing the PUF.

**Definition 5 (Data Remanence Experiment).** *Consider a device that overwrites the memory used by the PUF with some known data. Let $v_{\text{nom}}$ be the nominal supply voltage of the device. Let $\boldsymbol{M}_{\text{PUF}}$ be the PUF state (Definition 3) and $\boldsymbol{M}_{\text{Init}}$ be the initial state of the device memory. Further, let* Dev *be the algorithm describing the device behavior (Definition 4) with respect to some start-up state $\boldsymbol{M}_t$ (Definition 2). The* data remanence experiment $X = \mathsf{DRE}(\boldsymbol{M}_{\text{Init}}, t, Q)$ *is as follows:*

1. *Set the memory content of the device to $\boldsymbol{M}_{\text{Init}}$*
2. *Temporarily set the supply voltage of the device to 0 V for time $t$ and then set it back to $v_{\text{nom}}$*
3. *Send the query $Q$ to the device and observe its response $X = \mathsf{Dev}(\boldsymbol{M}_t, Q)$*

### 3.3 Finder Algorithm

Another building block of our attack is the finder algorithm, which recovers the PUF state based on the device behavior observed in a series of data remanence experiments.

**Definition 6 (Finder Algorithm).** *Let* $\boldsymbol{M}_{t_{i+1}}$ *and* $\boldsymbol{M}_{t_i}$ *be two start-up states that consist of* $n$ *bits and that differ in at most* $k < n$ *bits, i.e., the Hamming distance* $\mathsf{dist}\left(\boldsymbol{M}_{t_i}, \boldsymbol{M}_{t_{i+1}}\right) \leq k$*. Further, let* $X_{i+1} = \mathsf{Dev}\left(\boldsymbol{M}_{t_{i+1}}, Q\right)$ *for some arbitrary device query* $Q$*. A* finder algorithm *is a probabilistic polynomial time algorithm* $\mathsf{Finder}\left(\boldsymbol{M}_{t_i}, Q, X_{i+1}\right)$ *that returns* $\boldsymbol{M}_{t_{i+1}}$*.*

The finder is most efficient when $\mathsf{dist}\left(\boldsymbol{M}_{t_i}, \boldsymbol{M}_{t_{i+1}}\right)$ is minimal, ideally one. In this case, Finder can recover an unknown $n$-bit start-up state $\boldsymbol{M}_{t_{i+1}}$ from $\boldsymbol{M}_{t_i}$ and $X_{i+1}$ by performing a simple exhaustive search with linear complexity in $n$. However, $\mathsf{dist}\left(\boldsymbol{M}_{t_i}, \boldsymbol{M}_{t_{i+1}}\right)$ is typically larger than one since multiple SRAM cells may have similar remanence decay times or may be metastable (i.e., take a random value) [9,23,1,14]. In the worst case, where up to $k$ bits have changed in a start-up state with $n$ bits, a trivial finder performing an exhaustive search may require up to $\sum_{\ell=1}^{k} \binom{n}{\ell}$ steps. Observe that $n$ typically is a fixed system parameter while $k$ strongly depends on the quality of the equipment used for controlling the remanence decay in the SRAM. As we discuss in Section 6, the adversary can reduce $k$ significantly by using more accurate equipment and he may also use a Finder algorithm that is more efficient than the trivial approach.

### 3.4 Details of the Attack

The attack is detailed in Algorithm 1 and works as follows: The adversary chooses an arbitrary device query $Q$ (Step 1) and records the response $X_{\mathrm{PUF}}$ generated by the device using the PUF state $\boldsymbol{M}_{\mathrm{PUF}}$ (Step 2). Then, the adversary performs a series of DRE experiments (Definition 5) where he slightly increases the power-off time $t_i$ used in each experiment (Steps 3 and 4).[4] This way, he obtains a sequence of device responses $X_1, \ldots, X_f$ to the same query $Q$ generated by the device using the start-up states $\boldsymbol{M}_{t_1}, \ldots, \boldsymbol{M}_{t_f}$, respectively, where $\mathsf{dist}\left(\boldsymbol{M}_{t_i}, \boldsymbol{M}_{t_{i+1}}\right)$ for all $1 \leq i \leq (f-1)$ is upper bounded by some value $k$. Observe that $\boldsymbol{M}_{t_0} = \boldsymbol{M}_{\mathrm{Init}}$ is the initial state (Definition 1) and $\boldsymbol{M}_{t_f} = \boldsymbol{M}_{\mathrm{PUF}}$ is the PUF state (Definition 3) of the SRAM. Next, the adversary uses the Finder algorithm (Definition 6) to iteratively recover $\boldsymbol{M}_{\mathrm{PUF}}$ from the device responses observed in Steps 3 to 4. Specifically, starting from the known initial state $\boldsymbol{M}_{t_0} = \boldsymbol{M}_{\mathrm{Init}}$, the adversary iteratively recovers each $\boldsymbol{M}_{t_{i+1}}$ from $\boldsymbol{M}_{t_i}$ and $X_{i+1}$ until he arrives at the PUF state $\boldsymbol{M}_{t_f} = \boldsymbol{M}_{\mathrm{PUF}}$ (Step 6).

**Theorem 1 (Success of the Attack).** *The attack in Algorithm 1 successfully recovers the PUF state* $\boldsymbol{M}_{\mathrm{PUF}}$*. The worst case complexity of the attack when using a trivial* Finder *algorithm (Definition 6) is* $f \cdot \sum_{\ell=1}^{k} \binom{n}{\ell}$*, where* $f$ *is the number of* DRE *experiments (cf. Definition 5),* $n$ *is the size of the SRAM and* $k$ *is the maximum Hamming distance of the start-up states* $\boldsymbol{M}_{t_i}$ *and* $\boldsymbol{M}_{t_{i+1}}$ *used by the device in two consecutive* DRE *experiments for all* $1 \leq i \leq (f-1)$*.*

Note that the complexity of the attack strongly depends on the value of $k$, which highly depends on the accuracy of the equipment and the method used to control

[4] An adversary using the voltage-based approach would gradually lower the supply voltage (for a fixed amount of time) instead of increasing the power-off time.

**Algorithm 1** Extracting the PUF State of an SRAM PUF-Enabled Device

Consider a device that writes a known initial state $\boldsymbol{M}_{\text{Init}}$ (Definition 1) to the SRAM after it has been used as a PUF. Let $t_\infty$ be the decay time (cf. Definition 3) of the SRAM and let $\Delta t$ be the difference between the power-off times used in two consecutive $\mathsf{DRE}$ experiments (cf. Definition 5). Let $i, f \in \mathbb{N}$ be indices. The attack works as follows:

1. Fix an arbitrary device query $Q$
2. Record $X_{\text{PUF}} = \mathsf{DRE}\left(\boldsymbol{M}_{\text{Init}}, t_\infty, Q\right)$
3. Set $i \leftarrow 0$ and $t_0 = 0$
4. Repeat:
   (a) Set $i \leftarrow i + 1$
   (b) Set $t_i = t_{i-1} + \Delta t$
   (c) Record $X_i = \mathsf{DRE}\left(\boldsymbol{M}_{\text{Init}}, t_i, Q\right)$
   (d) Stop when $X_i = X_{\text{PUF}}$ and set $f = i$
5. Set $i \leftarrow 0$ and $\boldsymbol{M}_{t_0} = \boldsymbol{M}_{\text{Init}}$
6. Repeat:
   (a) Set $i \leftarrow i + 1$
   (b) Compute $\boldsymbol{M}_{t_i} = \mathsf{Finder}\left(\boldsymbol{M}_{t_{i-1}}, Q, X_i\right)$
   (c) Stop when $i = f$
7. Return $M_{t_f}$

the remanence decay in the SRAM. Typical values are $k = 0.0485 \cdot n$ for the time-based approach and $k = 0.0285 \cdot n$ for the voltage-based approach (cf. Section 4). Moreover, in our experiments we observed a decay time of $t_\infty = 2,000\ \mu s$ and used $\Delta t = 1\ \mu s$, resulting in $f = \lceil 2,000\ \mu s / 1\ \mu s \rceil = 2,000$.

*Proof (Theorem 1).* It follows from Definition 5 that $X_{\text{PUF}} = \mathsf{Dev}\left(\boldsymbol{M}_{t_\infty}, Q\right)$ and from Definition 3 that $\boldsymbol{M}_{t_\infty} = \boldsymbol{M}_{\text{PUF}}$. Hence, in Step 2, $X_{\text{PUF}}$ is the response of the device using the PUF state. Furthermore, it follows from Definition 5 that $X_i = \mathsf{Dev}\left(\boldsymbol{M}_{t_i}, Q\right)$ in Step 4(c). Hence, after Step 5 we have obtained a sequence of device responses $X_0, \ldots, X_f$ that correspond to the memory states $\boldsymbol{M}_{t_0}, \ldots, \boldsymbol{M}_{t_f}$. Due to the different decay times of the individual SRAM cells and the metastability in the SRAM, two memory states $\boldsymbol{M}_{t_i}$ and $\boldsymbol{M}_{t_{i+1}}$ differ in at most $k < n$ bits. Hence, $\mathsf{dist}\left(\boldsymbol{M}_{t_i}, \boldsymbol{M}_{t_{i+1}}\right) \leq k$ and it follows from Definition 6 that $\mathsf{Finder}\left(\boldsymbol{M}_{t_{i-1}}, Q, X_{i-1}\right) = \boldsymbol{M}_{t_i}$ in Step 6(b). By definition it holds that $\boldsymbol{M}_{t_0} = \boldsymbol{M}_{\text{Init}}$ and by induction over $i$ it follows that $\boldsymbol{M}_{t_f} = \boldsymbol{M}_{\text{PUF}}$ in Step 7.

It remains to show the complexity of the attack. In the worst case, $\mathsf{Finder}$ performs an exhaustive search over all $\sum_{\ell=1}^{k} \binom{n}{\ell}$ possible positions of the up to $k$ bits in which the $n$-bit state $\boldsymbol{M}_{t_{i+1}}$ may differ from $\boldsymbol{M}_{t_i}$. This means that in the worst case $\mathsf{Finder}$ must verify $\sum_{\ell=1}^{k} \binom{n}{\ell}$ guesses to find the correct memory state $\boldsymbol{M}_{t_i}$ in each of the $f$ iterations of Step 6(b). This leads to an overall attack complexity of $f \cdot \sum_{\ell=1}^{k} \binom{n}{\ell}$, which finishes the proof. □

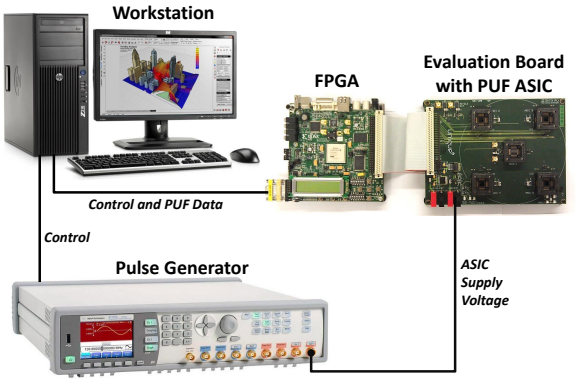


Fig. 1: Test setup with Xilinx Virtex 5 FPGA, ASIC evaluation board with one PUF ASIC, Agilent 81150 pulse generator and workstation.

## 4 Experimental Validation of the Attack

Our attack requires that only a small number of SRAM cells transition from the known (initial) state to the PUF state in two consecutive DRE experiments. This number is mainly affected by two factors: (1) the accuracy of the equipment used to control the remanence decay in the SRAM during the attack and (2) the number of SRAM cells that are metastable, i.e., that take a random state. In this section, we investigate the impact of both factors on the remanence decay in the SRAM PUFs implemented in two 65 nm CMOS ASICs. Our evaluation focuses on the time-based approach to control the remanence decay and concludes with some preliminary results on the voltage-based approach.

### 4.1 Test Setup

Our analysis is based on data obtained from two ASICs that have been manufactured in TSMC 65 nm CMOS technology within an Europractice multi-project wafer run. Each ASIC implements four different SRAM PUF instances, each using 8 kBytes of SRAM. The test setup consists of an ASIC evaluation board, a Xilinx Virtex 5 FPGA, an Agilent 81150 pulse/function/arbitrary pulse generator and a workstation (Figure 1). The evaluation board allows controlling the ASIC supply voltage using an external power supply. In each experiment, we wrote a pre-determined bit pattern (i.e., all ones) to the SRAM, used the pulse generator to deliver a temporary voltage drop with precisely controlled width and amplitude and finally read back the memory contents of the SRAM. The rated accuracy of the pulse generator is a temporal resolution of 5 ns and an amplitude resolution of 25 mV. To accelerate the remanence decay process, we did not place any coupling capacitors between the pulse generator's output and

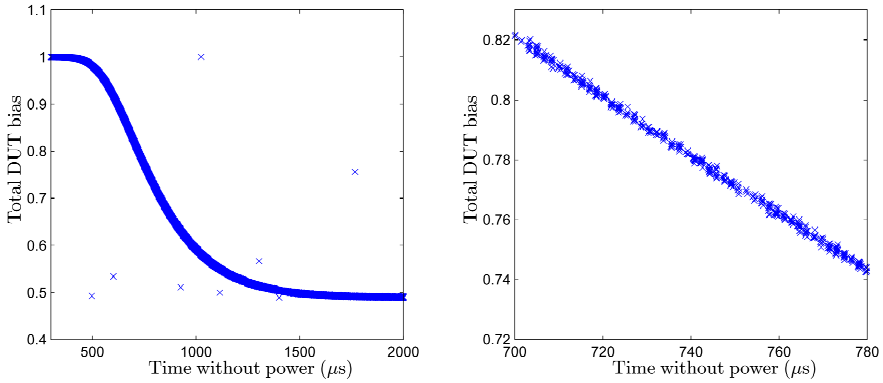


Fig. 2: A Chip-Scale View of Time-based Remanence Decay

the ASIC's supply voltage input. The interaction with the evaluation board and the ASICs is performed by the FPGA, which is connected to a workstation that controls the PUF evaluation and the pulse generator. Further, the workstation is used to process and store the data obtained from the ASICs. All tests have been performed at room temperature (approx. 25°C) in an air conditioned laboratory.

### 4.2 Chip-Scale Modeling

The purpose of this experiment was to observe and to reproduce the decay behaviour reported in [23] and gauge its stability and reproducibility for the SRAM PUF. A series of data remanence experiments with an initial state $\boldsymbol{M}_{\text{Init}}$ consisting of only ones was performed. Each experiment was repeated 10 times with 1,000 different power-off times $t$ between 300 $\mu$s and 2,000 $\mu$s. During the power-off time the supply voltage was set to 0 V. After each experiment we measured for each SRAM cell the probability that it still stores the value we wrote to it before the power cycle. We call this probability the *bias* of the cell.

Our results are depicted in Figure 2. The graph on the right represents a zoomed-in portion of the graph on the left. In both graphs, the x-axis corresponds to the total time the ASIC was without power, while the y-axis corresponds to the mean bias over all SRAM cells. Each cross in the graph corresponds to a single experiment. As shown in the left graph, the average bias over all SRAM cells decays very reliably from 1 to the expected 0.5 [1,14] during the course of 2 ms. As the detailed view in the graph on the right shows, there was a small variation in the measured bias between identical experiments, which was either due to the physical limitations of our test setup or due to the inherent metastability of some of the SRAM cells.

### 4.3 Bit-Scale Modeling

The next experiment investigates whether the individual SRAM cells have different transition times, which is required in our attack. With the *transition time*

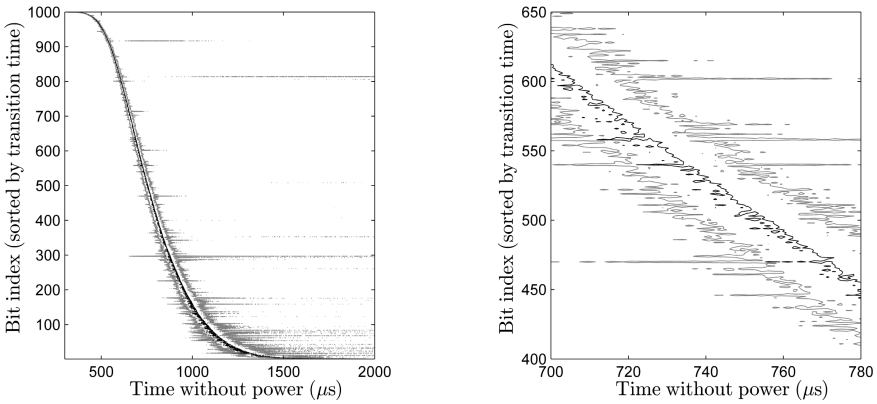


Fig. 3: A Bit-Scale View of Time-based Remanence Decay

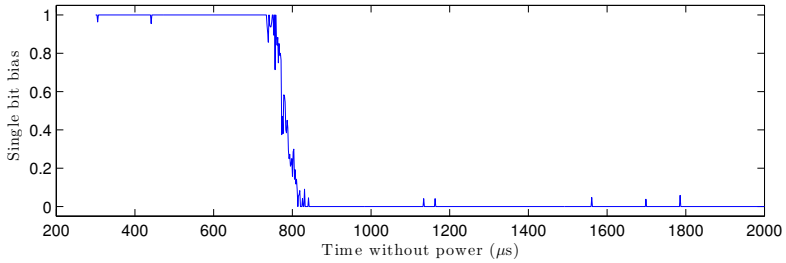


Fig. 4: A Close-up Look at a Single Bit

of an SRAM cell we mean the point in time where the cell loses the value that has been written to it and reverts to its PUF state. Based on the results of the previous experiment, we estimated the bias of each SRAM cell over time.

Figure 3 displays a 2-D contour plot of the cell-level behaviour of the SRAM PUF. Again, the graph on the right represents a zoomed-in portion of the graph on the left. Each horizontal row in the graph corresponds to the bias of a single SRAM cell selected out of 500 representative cells whose final bias was close to zero.[5] For the purpose of legibility, the cells were sorted in the graph by their transition time. The left and right gray lines on the graph correspond to times when the bias of each bit is one and zero, respectively, while the black line corresponds to the time when the bias of each bit is 0.5. A detailed look at the evolution of the bias of a single bit over time is shown in Figure 4.

As shown in Figure 3, each individual SRAM cell has a different remanence decay time surrounded by a short period of metastability in which the cell may enter both states. The median metastability period measured was 56 $\mu$s and the

[5] We only selected cells with a final bias close to zero since the cells with a final bias close to one will not show any decay behavior in our experiment where we wrote a logical one to all memory cells before the power cycle.

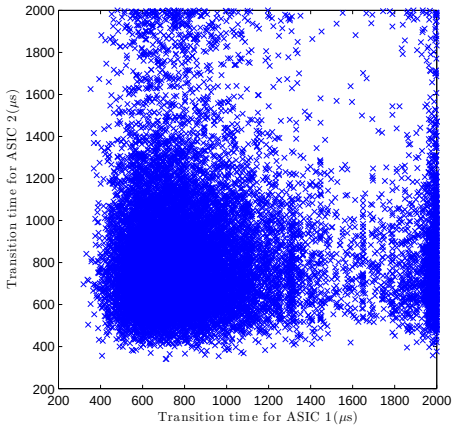


Fig. 5: Correlation Between the Transition Time in Two Different Devices

worst-case metastability rate was 4.83 %. In general, the maximum size of a PUF that can be attacked using our methodology is limited by the metastability, as we discuss further in Section 6.2.

### 4.4 Cross-Device Comparison

Next we investigated whether the transition times of the SRAM cells in one device allow to infer some information on the transition times of the SRAM cells in another device. A second goal of this experiment was to get a first impression of whether the transition times in SRAM cells could be used to identify individual SRAM chips, an idea we discuss in Section 6. In this experiment, we measured the bias over time and the transition times of each SRAM cell in both ASICs. Again, we considered only cells whose PUF state is zero.

The results are shown in Figure 5. Each cross in the graph corresponds to the bias of a single SRAM cell. The x-coordinate of each point is the transition time of the SRAM cell on the first ASIC, while the y-coordinate is the transition time of the same SRAM cell on the second ASIC. As Figure 5 shows, the transition times of the two ASICs are virtually uncorrelated, which we confirmed by computing the normalized cross correlation $\rho$ of both data sets, which is $\rho = -0.053$. Our results are in line with the findings by Holcomb et al. [11] who also suggest using the remanence decay behaviour as a source of unique information to identify individual devices.

### 4.5 Time-Based vs. Voltage-Based Attacks

As discussed in Section 3.1, there are two ways of controlling the remanence decay in an SRAM: Varying the *time* for which the device is held without power and varying the *voltage* experienced by the device. It has been shown [23] that

Table 1: Preliminary Comparison of Voltage-Based and Time-Based Remanence

| **Remanence control** | **Voltage-based** | **Time-based** |
|---|---|---|
| **Bits stable at 1** | 79.86 % | 79.80 % |
| **Bits stable at 0** | 17.29 % | 15.37 % |
| **Metastability rate (worst case)** | 2.85 % | 4.83 % |

the voltage-based approach is less sensitive to temperature variations, making it potentially more effective in an attack than the time-based approach. While the pulse generator we used in our experiments had a very sensitive temporal resolution of 5 ns, it had an amplitude resolution of only 25 mV, which was not sufficient to carry out a complete attack using the voltage-based approach. However, we still present preliminary results based on a single stable voltage and a single time period.

Our results are summarized in Table 1, which shows that using the voltage-based approach results in a significantly lower metastability rate than using the time-based approach. This means that a voltage-based attack will potentially be effective in situations where the time-based attack will fail. An interesting observation is that the set of metastable SRAM cells in both experiments was quite different, which indicates that most of the inaccuracies in our experiments are due to the limitations of our test setup and not due the physical properties of the SRAM PUF itself.

## 5 Practical Validation of the Attack

To investigate the effectiveness of our attack in a practical setting, we created a standard implementation of an SRAM PUF-based authentication scheme. This scheme uses a standard secret-key-based challenge-response protocol and derives the underlying key from the PUF response using a basic repetition code [3].[6]

More detailed, during the enrollment of the device, the memory addresses of those 128 SRAM bytes whose PUF state is highly biased (i.e., that have a Hamming weight of 0, 1, 7 or 8) are stored as the public helper data, each representing one bit of the secret key stored in the PUF. The key is reconstructed from the PUF as follows: The 128 SRAM bytes whose addresses are stored in the helper data are read from the SRAM and the value of each bit in the key is set as the result of a simple majority voting over all bits in the respective byte. The resulting secret key $K$ is then used in the secret key-based challenge-response protocol, i.e. $X = \mathsf{MAC}_K(Q)$, where $\mathsf{MAC}$ is a message authentication code.

The attack is as in Section 3.4. However, we use an optimized $\mathsf{Finder}$ algorithm (Definition 6) that only searches for key candidates with a Hamming distance less

[6] We omit the linear encoding used in [3] and the privacy amplification typically used in PUF-based key storage since it has no effect on our attack.

than 10 bits from the previous key, which significantly improves the performance of the attack compared to the trivial Finder described in Section 3.3.

The overall running time of the attack is estimated as $2^{53.6}$ MAC operations. Considering that modern CPUs can perform $2^{31}$ AES operations per second, the total cost of the attack when using an AES-based MAC is $2^{22.6}$ CPU-seconds, or approximately two CPU-months. The attack can easily be parallelized by testing multiple attempts or multiple key candidates simultaneously, making it even more practical for moderately-funded adversaries.

## 6 Impact of Our Attack and Countermeasures

### 6.1 Impact

Our results in Section 4 show that by carefully controlling the power-off times of the SRAM PUF, one can reliably control the number of metastable bits as required by the attack described in Section 3. Our current best results show that the average number of metastable SRAM cells can be limited to about 1 % of the total memory size. This means that, even if we use the trivial finder algorithm discussed in Section 3.3, common lab equipment and the less effective time-based approach to control the remanence decay in the SRAM, we can recover the PUF response of a 216-bit SRAM PUF by making at most $2^{64}$ calls to the Dev algorithm (cf. Definition 4). Using the voltage-based approach with the same finder algorithm and equipment as in the time-based approach, we can extract the response of a 315-bit SRAM PUF in the same time. Further, our results in Section 5 show that, depending on the post-processing of the PUF responses, our attack can also be applied to systems using larger PUFs. Hence, it is problematic to overwrite the memory of an SRAM PUF with a known value, which, however, is required when the PUF memory is also used for other purposes, as suggested in many prior works [30,6,25,5,13,16,15]. This particularly holds for resource-constrained devices with only small amounts of SRAM, such as RFIDs or medical implants [30,25,5], where SRAM PUFs without shared memory are impractical.

### 6.2 Improving the Attack

One approach to lower the complexity of our attack is using more accurate equipment that allows a very precise control of the remanence decay in the SRAM using the voltage-based approach, which limits the number of metastable bits and the complexity of the finder algorithm (cf. Section 3.3).

Furthermore, several optimizations of the finder algorithm are possible: The order in which the individual SRAM cells transition from their initial state to their PUF state is different for the time-based and the voltage-based approach (cf. Section 4.5). Further, in some scenarios the adversary may be able to control the initial state of the SRAM. This results in four different ways to observe the decay behavior of each SRAM cell and allows the adversary to chose the way

with the lowest metastability rate for his attack, which can significantly reduce the complexity of the naive finder algorithm (cf. Section 3.3).

Another approach to improve the complexity of the finder algorithm is to take advantage of the algorithms used by the device to process the PUF responses (cf. Section 5). These algorithms typically include an error correction mechanism [3] to handle errors in the PUF response that come from environmental variations affecting the underlying physical object. Due to this error correction the device response changes only when the error correction mechanism fails. Hence, the finder algorithm needs to consider only one single candidate of each codeword class. This can either be done explicitly by considering the structure of the error correcting code or by casting the problem as an optimization problem and using an optimizer [22].

### 6.3 Countermeasures

There are several countermeasures that prevent our attack by breaking the underlying assumptions but that are impractical in low-resource scenarios such as RFIDs and sensors [30,25,5]. One approach to prevent the attack described in Section 3 is using an additional memory that can only be accessed by the PUF. However, this contradicts the idea of using the existing memory of the device and significantly increases the implementation costs. Another approach is to wait until any value stored in the memory has decayed before reading the PUF response. However, this requires the device to have some notion of time and significantly increases the boot time, which is problematic in many applications. Further, the attack can be prevented by designing the algorithms processing the PUF response such that the device behavior for different start-up states is indistinguishable by the adversary. However, this seems to imply the use of complex cryptographic primitives such as anonymous authentication schemes that typically exceed the capabilities of resource-constrained devices for which SRAM PUFs with shared memory have been proposed [30,25,5].

## 7 Constructive Use of Data Remanence Decay

### 7.1 Device Authentication Based on SRAM Remanence Decay

The remanence decay behavior can be used to authenticate an SRAM to some verifier. Specifically, using the same approach as in our attack, a verifier could force the SRAM into a partially reverted state by writing some value to the SRAM and then powering the device off for a carefully controlled amount of time. Since the verifier knows the (secret) PUF state of the SRAM and the decay behavior of the genuine device, he can determine the partially reverted SRAM state of the device and check whether it matches the expected state of the SRAM to be authenticated. Care must be taken that this additional functionality does not expose the device to our attack, for example by requiring that the verifier successfully authenticates to the device before he can access the SRAM.

Note that it is much more difficult to clone such an SRAM PUF since the clone must emulate the SRAM decay behavior, which requires the clone to contain a time-keeping mechanism, raising its costs. Our results suggest that for an SRAM of size $n$ bits there are $n \log n$ bits of entropy encoded in the *order* in which individual SRAM cells revert to their PUF state. However, further evaluations are needed to asses the practicality of this approach, in particular the temperature dependency and the effect of aging on the decay behavior of SRAM must be investigated.

### 7.2 Improving the TARDIS Time-Keeping Algorithm

The use of SRAM remanence decay has recently been proposed as a time-keeping mechanism for clock-less low-power devices, such as passive RFID tags [23]. This mechanism, called TARDIS, allows a clock-less device to estimate how much time has passed since its last power-down and aims to impede oracle attacks. TARDIS consists of two main elements: The Init algorithm which sets all SRAM cells to a fixed value (all ones) and the Decay algorithm which determines how long the device has been without power based on the number of ones that are still stored in the SRAM. Observe that the Init algorithm requires to write a one to each cell of the SRAM, while the Decay algorithm must read the value of each cell while the device is booting. These two operations consume a non-negligible amount of power and add an additional 15.2 ms to the start-up time of the device.

Our observations on the behaviour of remanence decay can be used to dramatically improve the performance of the TARDIS system. As our results show, the transition time of each bit is uniquely determined by its individual data remanence voltage (DRV). By profiling the SRAM in an offline phase, we can thus determine the *order* in which the SRAM cells return to their PUF state and store this ordering in the non-volatile memory of the device. Now, if we observe that a certain group of bits has reverted to its PUF state, we immediately know that all bits which have a shorter transition time have also returned to their PUF state. Similarly, if we observe that a certain group of bits is still in its initial state, we immediately know that all bits that have a longer transition time are also still in their initial state. Knowing this ordering, we can replace the linear-time Decay algorithm of [23] with the well known binary search algorithm that takes logarithmic time. To deal with metastability, the algorithm should sample not only one but a group of bits for each transition time period.

If the device needs to detect only whether or not the *entire* SRAM has returned to its PUF state, another improvement is possible that dramatically decreases the running time of both the Init and the Decay algorithms from linear time to constant time. In this case, both algorithms need only to access those SRAM cells that are known to be the *last* to revert to the PUF state.

Since most of the applications described in [23] can be adapted to use these improvements, our results enhance the applicability of the TARDIS system to practical scenarios. We stress that the SRAM used by the TARDIS scheme cannot be used as an SRAM PUF since its content is well-known in this case.

## 8 Related Work

While the impact of remanence decay on the randomness that can be extracted from SRAM cells and the reliability of SRAM PUFs has been discussed in the literature [29,26,10,27,11], it has never been used as a side channel to attack SRAM PUFs. In fact, there are only a few papers [12,21] discussing side channel attacks in the context of PUFs. However, these papers mainly focus on the side channel leakage of the algorithms processing the PUF response and only vaguely discuss potential side channels of PUFs. The impact of abnormal operating conditions on the unpredictability and the reliability of memory-based PUFs has been evaluated [1,14] but no results on fault injection attacks on PUFs have been reported. In contrast, to the best of our knowledge, we present the first cloning attack that injects faults into the SRAM PUF and uses the data remanence effects in SRAM as a side channel to recover the (secret) PUF response.

Data remanence in DRAM has been used to extract security-sensitive data from the random access memory of PCs and workstations [8]. While these attacks aim to recover some data that has been written to an unprotected memory, the goal of our attack is to recover the start-up pattern of an SRAM PUF that is typically protected by some kind of access control mechanism.

## 9 Conclusion

We demonstrated a simple non-invasive cloning attack on SRAM PUFs using remanence decay as a side-channel and validated its feasibility against two SRAM PUF implementations in two 65 nm CMOS ASICs. Our attack and evaluation is general and can be optimized for concrete systems. Our evaluation results show that even without optimizations, attacks on small SRAM PUFs are feasible using common lab equipment. We discussed countermeasures against our attack and suggest using remanence decay to improve the cloning-resistance of SRAM PUFs. As a contribution of independent interest, we showed how our evaluation results can be used to improve the performance of TARDIS [23], a recently proposed time-keeping mechanism for clock-less devices.

We mainly focused on the time-based approach to control the data remanence decay in the SRAM. We are currently evaluating the voltage-based approach that seems to be more promising than the time-based approach and may help to increase the performance and efficiency of our attack. Other directions for future work include the design of non-trivial finder algorithms that, e.g., exploit the properties of the algorithms used by the device processing the PUF response.

**Acknowledgements.** We thank Ünal Kocabaş for preparing the lab experiments in the first phase of this work. The development and manufacturing of the PUF ASIC used in this work has been supported by the European Commission under grant agreement ICT-2007-238811 UNIQUE.

## References


1. Bhargava, M., Cakir, C., Mai, K.: Comparison of bi-stable and delay-based physical unclonable functions from measurements in 65nm bulk CMOS. In: Custom Integrated Circuits Conference (CICC). pp. 1–4. IEEE (2012)
2. Biham, E., Shamir, A.: Differential fault analysis of secret key cryptosystems. In: Advances in Cryptology (CRYPTO). LNCS, vol. 1294, pp. 513–525. Springer (1997)
3. Bösch, C., Guajardo, J., Sadeghi, A.R., Shokrollahi, J., Tuyls, P.: Efficient helper data key extractor on FPGAs. In: Cryptographic Hardware and Embedded Systems (CHES). LNCS, vol. 5154, pp. 181–197. Springer (2008)
4. Eichhorn, I., Koeberl, P., van der Leest, V.: Logically reconfigurable PUFs: Memory-based secure key storage. In: ACM Workshop on Scalable Trusted Computing (ACM STC). pp. 59–64. ACM (2011)
5. Guajardo, J., Asim, M., Petković, M.: Towards reliable remote healthcare applications using combined fuzzy extraction. In: Towards Hardware-Intrinsic Security. pp. 387–407. Information Security and Cryptography, Springer (2010)
6. Guajardo, J., Kumar, S.S., Schrijen, G.J., Tuyls, P.: Physical unclonable functions and public-key crypto for FPGA IP protection. In: Field Programmable Logic and Applications (FPL). pp. 189–195. IEEE (2007)
7. 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 (ISCAS). pp. 3186–3189. IEEE (2008)
8. Halderman, J.A., Schoen, S.D., Heninger, N., Clarkson, W., Paul, W., Calandrino, J.A., Feldman, A.J., Appelbaum, J., Felten, E.W.: Lest we remember: Cold-boot attacks on encryption keys. Communications of the ACM 52(5), 91–98 (2009)
9. Holcomb, D., Burleson, W., Fu, K.: Initial SRAM state as a fingerprint and source of true random numbers for RFID tags. In: Workshop on RFID Security (RFIDSec) (2007)
10. Holcomb, D., Burleson, W.P., Fu, K.: Power-Up SRAM state as an identifying fingerprint and source of true random numbers. IEEE Transactions on Computers 58(9), 1198–1210 (2009)
11. Holcomb, D.E., Rahmati, A., Salajegheh, M., Burleson, W.P., Fu, K.: DRV-fingerprinting: Using data retention voltage of SRAM cells for chip identification. In: Hoepman, J.H., Verbauwhede, I. (eds.) Radio Frequency Identification. Security and Privacy Issues. LNCS, vol. 7739, pp. 165–179. Springer (2013)
12. Karakoyunlu, D., Sunar, B.: Differential template attacks on PUF enabled cryptographic devices. In: Workshop on Information Forensics and Security (WIFS). pp. 1–6. IEEE (2010)
13. Kardas, S., Kiraz, M.S., Bingol, M.A., Demirci, H.: A novel RFID distance bounding protocol based on physically unclonable functions. In: Radio Frequency Identification: Security and Privacy Issues (RFIDSec). LNCS, Springer (2011)
14. Katzenbeisser, S., Kocabaş, U., 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 (CHES). LNCS, vol. 7428, pp. 283–301. Springer (2012)
15. Koeberl, P., Li, J., Maes, R., Rajan, A., Vishik, C., Wójcik, M.: Evaluation of a PUF device authentication scheme on a discrete 0.13$\mu$m SRAM. In: International Conference on Trusted Systems (INTRUST). LNCS, vol. 7222, pp. 271–288. Springer (2012)

16. Koeberl, P., Li, J., Rajan, A., Vishik, C., Wu, W.: A practical device authentication scheme using SRAM PUFs. In: Conference on Trust and Trustworthy Computing (TRUST). LNCS, vol. 6740, pp. 63–77. Springer (2011)
17. Kumar, S.S., Guajardo, J., Maes, R., Schrijen, G.J., Tuyls, P.: Extended abstract: The butterfly PUF protecting IP on every FPGA. In: Workshop on Hardware-Oriented Security (HOST). pp. 67–70. IEEE (2008)
18. van der Leest, V., Schrijen, G.J., Handschuh, H., Tuyls, P.: Hardware intrinsic security from D flip-flops. In: ACM Workshop on Scalable Trusted Computing (ACM STC). pp. 53–62. ACM (2010)
19. Lim, D., Lee, J.W., Gassend, B., Suh, E.G., van Dijk, M., Devadas, S.: Extracting secret keys from integrated circuits. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 13(10), 1200–1205 (2005)
20. Maes, R., Tuyls, P., Verbauwhede, I.: Intrinsic PUFs from flip-flops on reconfigurable devices. In: Benelux Workshop on Information and System Security (2008)
21. Merli, D., Schuster, D., Stumpf, F., Sigl, G.: Side-channel analysis of PUFs and fuzzy extractors. In: Trust and Trustworthy Computing (TRUST). LNCS, vol. 6740, pp. 33–47. Springer (2011)
22. Oren, Y., Renauld, M., Standaert, F.X., Wool, A.: Algebraic Side-Channel attacks beyond the Hamming weight leakage model. In: Prouff, E., Schaumont, P. (eds.) Cryptographic Hardware and Embedded Systems (CHES). LNCS, vol. 7428, pp. 140–154. Springer (2012)
23. Rahmati, A., Salajegheh, M., Holcomb, D., Sorber, J., Burleson, W.P., Fu, K.: TARDIS: Time and remanence decay in SRAM to implement secure protocols on embedded devices without clocks. In: USENIX Security Symposium. pp. 36–52. USENIX Association (2012)
24. Roy, J.A., Koushanfar, F., Markov, I.L.: EPIC: ending piracy of integrated circuits. Computer 43(10), 30–38 (2010)
25. Sadeghi, A.R., Visconti, I., Wachsmann, C.: Enhancing RFID security and privacy by physically unclonable functions. In: Towards Hardware-Intrinsic Security. pp. 281–305. Information Security and Cryptography, Springer (2010)
26. Saxena, N., Voris, J.: We can remember it for you wholesale: Implications of data remanence on the use of RAM for true random number generation on RFID tags (RFIDSec 2009) (2009)
27. 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)
28. Su, Y., Holleman, J., Otis, B.P.: A digital 1.6 pJ/bit chip identification circuit using process variations. IEEE Journal of Solid-State Circuits 43(1), 69–77 (2008)
29. Tokunaga, C., Blaauw, D., Mudge, T.: True random number generator with a metastability-based quality control. IEEE Journal of Solid-State Circuits 43(1), 78–85 (2008)
30. Tuyls, P., Batina, L.: RFID-tags for anti-counterfeiting. In: Topics in Cryptology (CT-RSA). LNCS, vol. 3860, pp. 115–131. Springer (2006)