MEMORY-BASED HARDWARE-INTRINSIC SECURITY MECHANISMS FOR DEVICE AUTHENTICATION IN EMBEDDED SYSTEMS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Soubhagya Sutar

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

August 2020

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF DISSERTATION APPROVAL

*Dedicated to my parents, **Meena** and **Janmejay**, who taught me the values of sincerity and hard work and to my wife, **Nupur**, who gave me the gift of unconditional love*

ACKNOWLEDGMENTS

Pursuing a Ph.D. has been one of the most challenging but equally rewarding phases of my life. An incredible number of people have supported me on this journey and helped me achieve this long-cherished goal. First and foremost, I express my deepest gratitude towards my advisor, **Prof. Vijay Raghunathan**, for his intellectual guidance, active support, and constant encouragement throughout my entire academic life at Purdue University. I thank him for giving me the freedom to explore my areas of interest as a new Ph.D. student and sharing his valuable advice that helped me decide my dissertation topic. As my major advisor, he carefully stimulated ideas in me without explicitly prescribing the next steps, thereby helping me develop the thought process required of an independent researcher. His ability to present his ideas and work most articulately has always amazed me and helped me learn the skills of an effective presenter. I truly appreciate his support and understanding through the many ups and downs of my Ph.D. journey. Whether it is about a difficult paper rebuttal, finding an assistantship, or a personal issue, Prof. Raghunathan has always been a patient listener and helped me overcome the situation. I am extremely grateful to him for believing in me and allowing me to join his research group, where I could work towards addressing some of the most exciting and challenging research problems in embedded systems design.

I am also grateful for the immense support and understanding of the other members of my dissertation committee –

Prof. Anand Raghunathan, for his valuable advice that helped me get started as a new Ph.D. student, teaching me key concepts in embedded systems design, and giving me constructive feedback on my research and course projects,

Prof. Kaushik Roy, for sharing his invaluable insights and feedback on my research work, and

Prof. Shreyas Sen, for his helpful suggestions and constructive feedback towards improving key aspects of my dissertation.

While this journey has been challenging, it has certainly been enjoyable through the association of my lab-mates and friends. I sincerely thank my lab-mate, *Arnab*, who collaborated with me on several projects and guided me in my initial days in the group. I enjoyed our research discussions and appreciate the insights and ideas that he brought into them. I am extremely grateful to my lab-mate, *Hrishikesh*, who mentored me as a senior member of the group, motivated me and helped me build the skills required to pursue research. I am very grateful to the other members of my research group – *Younghyun* for his invaluable help with my first publication and *Woosuk* for helping me out with the TA duties. I would also like to thank my current lab-mates – *Soumendu* and *Malin* for giving me their precious support and allowing me to share my thoughts and feedback on their research work. Along this path, I also met several great people and made some very good friends. I would like to thank each one of them – *Krishna*, *Arnab*, *Priya*, *Hrishikesh*, *Shubham*, *Gowtham*, *Ashish*, and *Soumendu*, who gave me several fond memories over these years and made my life truly eventful.

Finally, I express my deepest gratitude to my parents, who are my first teachers and role models. I thank my mother, *Meena*, who taught me the importance of sincerity and hard work, deeply cared for me, and supported me selflessly all these years. I am very thankful to my father, *Janmejay*, who nurtured the curiosity in me about science and inspired me to pursue engineering as a career. I am also grateful to my grandparents and other family members, who have showered me with their blessings and love. Last but not the least, I dedicate this dissertation to my lovely wife, *Nupur*. I am truly humbled by her unconditional love, selfless support, and unwavering trust, all of which gave me the strength to embark upon this difficult and exciting journey.

TABLE OF CONTENTS

Page

Page

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

ABSTRACT

Sutar, Soubhagya Ph.D., Purdue University, August 2020. Memory-based Hardware-intrinsic Security Mechanisms for Device Authentication in Embedded Systems. Major Professor: Vijay Raghunathan.

The Internet-of-Things (IoT) is one of the fastest-growing technologies in computing, revolutionizing several application domains such as wearable computing, home automation, industrial manufacturing, *etc.* This rapid proliferation, however, has given rise to a plethora of new security and privacy concerns. For example, IoT devices frequently access sensitive and confidential information (*e.g.,* physiological signals), which has made them attractive targets for various security attacks. Moreover, with the hardware components in these systems sourced from manufacturers across the globe, instances of counterfeiting and piracy have increased steadily. Security mechanisms such as device authentication and key exchange are attractive options for alleviating these challenges.

In this dissertation, we address the challenge of enabling low-cost and low-overhead device authentication and key exchange in off-the-shelf embedded systems. The first part of the dissertation focuses on a hardware-intrinsic mechanism and proposes the design of two Physically Unclonable Functions (PUFs), which leverage the memory (DRAM, SRAM) in the system, thus, requiring minimal (or no) additional hardware for operation. Two lightweight authentication and error-correction techniques, which ensure robust operation under wide environmental and temporal variations, are also presented. Experimental results obtained from prototype implementations demonstrate the effectiveness of the design. The second part of the dissertation focuses on the application of these techniques in real-world systems through a new end-to-end authentication and key-exchange protocol in the context of an Implantable Medical

Device (IMD) ecosystem. Prototype implementations exhibit an energy-efficient design that guards against security and privacy attacks, thereby making it suitable for resource-constrained devices such as IMDs.

1. INTRODUCTION

Over the past few years, the Internet-of-Things (IoT) has pervaded all aspects of our daily lives. It has fundamentally altered the way we interact with our physical environment, thereby revolutionizing a number of application domains such as home automation, wearable computing, industrial manufacturing, *etc.* The IoT is also one of the fastest-growing technologies across all of computing and it is expected that there would 125 billion devices connected to it by the year 2030 [1]. However, this rapid proliferation coupled with the increasingly connected nature of these devices has given rise to a plethora of new security and privacy concerns, as shown in Fig. 1.1. One such example is data theft. Though preventing it has been a challenge right from the beginning of personal computing, the problem has assumed a completely new scale due to the sheer number of these smart devices with the ability to access potentially sensitive and confidential information (*e.g.,* banking credentials, physiological signals, *etc.*). Moreover, the diversity of these devices, ranging from tiny microcontrollers to large datacenter servers, makes it near impossible to employ a standard security solution. Another concern stems from the the cyber-physical nature of these devices, as a result of which, security attacks have physical consequences. For example, in 2015, attackers were able to get remote access to a Jeep Cherokee and kill some of its vital systems such as steering, braking, transmission, *etc* [2]. Also, there have been numerous security vulnerabilities identified in medical devices that could be exploited by attackers to cause serious harm to the patient [3, 4]. Finally, with the hardware components in these systems sourced from manufacturers across the globe, instances of counterfeiting and piracy have increased steadily. Counterfeit components not only have serious reliability implications, but also cause tremendous revenue loss to the electronics industry [5].

Fig. 1.1.: Security and Privacy Concerns in the IoT

1.1 Research Contributions and Dissertation Overview

The previous section highlighted several concerns associated with the rapid growth of the IoT in recent years. One way of addressing several of these, which the dissertation focuses on, is *device authentication*. Authentication, in general, refers to any process by which a system verifies the identity of a user who wishes to access it. In the context of the IoT, the user may refer to a client device trying to connect to a network while the system is a trusted authority and may refer to the network gateway. By ensuring that each of these devices is *authentic*, in other words, by performing device authentication, a secure network can be built, as shown in Fig. 1.2.

Traditionally, device authentication has been carried out using static techniques such as passwords, secret keys, public-key infrastructure (PKI), *etc*. However, these techniques have several shortcoming such as the need for user intervention and vulnerability to present-day sophisticated attacks. Most importantly, a large number of the IoT devices are low-cost in nature and lack the resources – compute, memory, en-

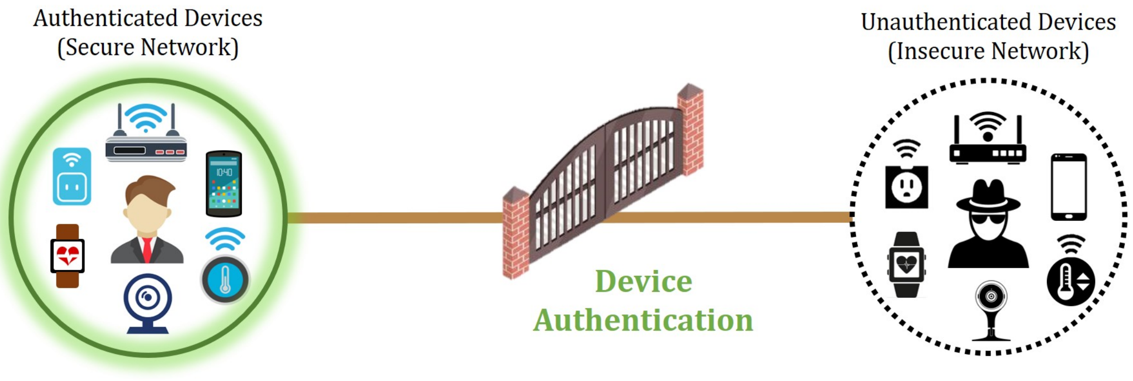
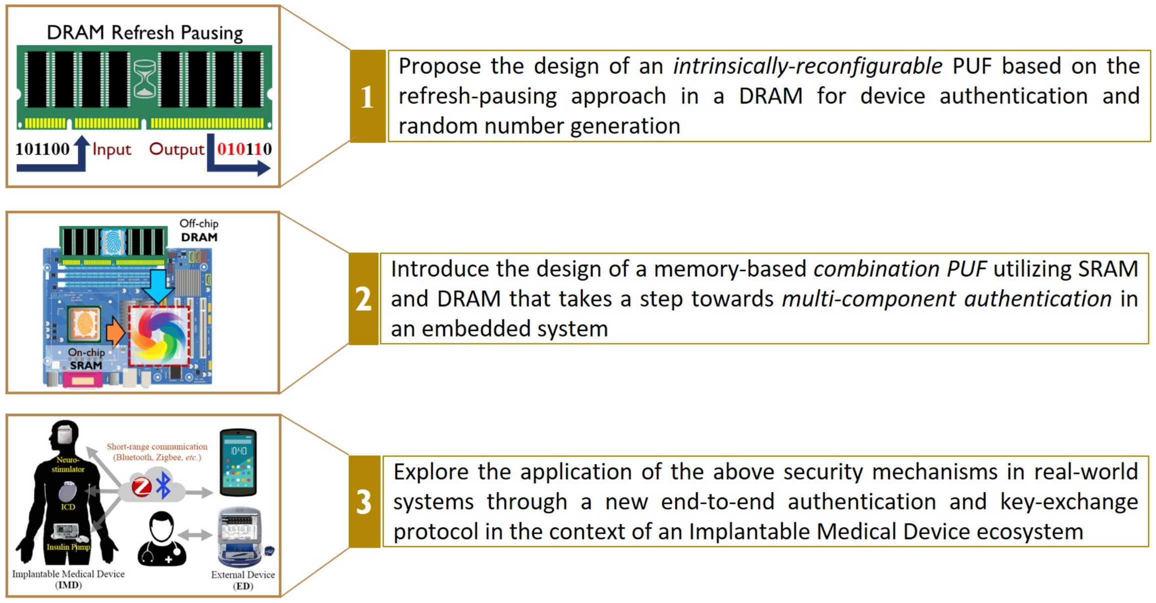


Fig. 1.2.: Building a secure IoT network through Device Authentication

ergy, *etc.*, which are required by some of these authentication techniques. All of these motivate us to look into alternate techniques for performing device authentication. One such technique is based on the use of hardware-intrinsic security mechanisms such as Physically Unclonable Functions (PUFs) [6]. PUFs exploit the random physical variations inherent to any manufacturing process to generate device-specific and unclonable fingerprints. These unique fingerprints can be used as the basis for the challenge-response mechanism for device authentication as well as for random number (*e.g.*, secure key) generation. While a large variety of PUF implementations have been proposed, this dissertation leverages memory-based PUFs due to the ubiquitous presence of memory in virtually every embedded system. Moreover, memory-based PUFs require minimal or no additional circuitry for their operation, giving them a distinct advantage over other PUF implementations. Hence, towards enabling low-cost and low-overhead device authentication in embedded systems, the dissertation makes the following research contributions, as shown in Fig. 1.3:

- The dissertation proposes the design of an *intrinsically-reconfigurable* PUF based on the refresh-pausing approach in a DRAM for device authentication and random number generation [7,8].



Propose the design of an *intrinsically-reconfigurable* PUF based on the refresh-pausing approach in a DRAM for device authentication and random number generation

Introduce the design of a memory-based *combination PUF* utilizing SRAM and DRAM that takes a step towards *multi-component authentication* in an embedded system

Explore the application of the above security mechanisms in real-world systems through a new end-to-end authentication and key-exchange protocol in the context of an Implantable Medical Device ecosystem

Fig. 1.3.: Research contributions made by this dissertation

- It introduces the design of a memory-based *combination PUF* utilizing SRAM and DRAM that takes a step towards *multi-component authentication* in an embedded system [9].
- It explores the application of the above security mechanisms in real-world systems through a new end-to-end authentication and key-exchange protocol in the context of an Implantable Medical Device ecosystem [10].

Next, we briefly introduce the work associated with each of the above contributions - a detailed discussion follows in the subsequent chapters.

1.1.1 An Intrinsically-Reconfigurable DRAM PUF for Device Authentication and Random Number Generation

Chapter 3 introduces the design of an intrinsically-reconfigurable PUF [7,8] based on the Dynamic Random Access Memory (DRAM). DRAM is used as the main memory in a large number of modern embedded systems due to its high density and low cost. As is well-known, DRAM cells must be refreshed periodically to preserve the stored data. This need for refresh operations also makes DRAM an attractive candidate for use as a PUF, especially for challenge-response-based authentication and random number generation.The key idea of a DRAM PUF based on refresh pausing is as follows (a more detailed explanation is given in Chapter 3). If data is stored in a large (say 64KB) block of DRAM cells and refresh operations to the entire block are paused for an extended amount of time (henceforth referred to as the refresh-pause interval), some of the DRAM cells in the block will lose their data. How many and which cells in the DRAM block lose their contents (for a given refresh-pause interval) is unique to a device and can, therefore, be used as the basis for implementing a PUF.

Prior approaches to DRAM-based PUFs suffer from several shortcomings such as low speed of authentication [11], non-applicability to commercial off-the-shelf (COTS) devices [12], and the need for power cycling the DRAM module prior to authentication [13]. Moreover, the near static nature of the response generation mechanism in some of the works [11, 13] makes them vulnerable to various security attacks [14]. To address these limitations, we propose a DRAM PUF, henceforth referred to as *D-PUF*, based on refresh pausing that not only supports a very large number of challenge-response pairs (CRPs) through the variation of different parameters, but is also *intrinsically reconfigurable*, *i.e.*, its challenge-response behavior can be substantially modified without the use of any additional circuitry. Hence, the PUF can be easily implemented in most off-the-shelf systems and provides considerable protection from various security attacks. We also use D-PUF to design a secure, low-overhead mechanism for performing device authentication. The mechanism operates robustly even in the presence of environmental and temporal variations. We implement D-PUF and our proposed authentication mechanism in a real system using off-the-shelf DRAM modules and evaluate it thoroughly. In particular, we demonstrate a 4.3X-6.4X reduction in authentication time, compared to previous work. Using controlled temperature and accelerated aging tests, we demonstrate the robustness of our authentication mechanism to temperature and aging effects. For a sample temperature range of 10°C to 60°C, we show that the mechanism achieves a 100% true-positive (successful authentication) rate and 0% false-positive rate. For a ten-month-old DRAM module, it also ensures a 100% true-positive rate and 0% false-positive rate. We also implement a True Random Number Generator (TRNG) using D-PUF by exploiting the Variable Retention Time (VRT) phenomenon in DRAMs and demonstrate that the generated random numbers pass all the tests specified in the NIST Statistical Test Suite.

1.1.2 Memory-Based Combination PUFs for Device Authentication in Embedded Systems

Current memory-based PUFs are constructed using a single memory component in the device, *i.e.,* based on a single entropy source. This means that the PUF represents the identity of the component and not that of the system. If the component is removed and transferred to a different system (invasive attack), the identity transfers over as well, which is undesirable. Moreover, the use of a single memory component makes it vulnerable to sophisticated non-invasive attacks [14]. To mitigate these concerns, it is desirable that the PUF be dependent on multiple system components (some of which may be more tightly integrated into the system than others), thereby performing *multi-component authentication*. Recent works [15,16] have tried to address a subset of these shortcomings. However, the choice of entropy sources used in these PUF designs renders them unsuitable for multi-component authentication. They also require the addition of custom hardware to the system and, hence, cannot be implemented using Commercial-Off-The-Shelf (COTS) systems.

As presented in Chapter 4, this dissertation overcomes these limitations by proposing the design of a memory-based combination PUF, henceforth referred to as *C-PUF* [9]. By tightly integrating heterogeneous memory technologies, *C-PUF* exhibits high entropy alongside an exponential number of CRPs, and takes the first step towards multi-component authentication in an embedded device. The heterogeneous nature of the entropy sources (memories) used and *C-PUF*'s ability to undergo *intrinsic reconfiguration* (ability to reconfigure the PUF at runtime without any additional hardware) protects it from various security attacks. *C-PUF* also features a lightweight authentication scheme to ensure robust operation (authentication) under wide environmental and temporal variations. We implement, demonstrate, and evaluate several fully-functional prototypes of *C-PUF* in a real system using two widely-used memory technologies, Static Random Access Memories (SRAMs) and Dynamic Random Access Memories (DRAMs). Extensive authentication tests performed across a wide temperature range (20°C – 55°C) and accelerated aging (12 months) achieved greater than 97.5% true-positive rate. The absence of any false-positives, even under an invasive attack, further highlights the effectiveness of the overall design.

1.1.3 A Lightweight End-to-End Authentication Protocol for Implantable Medical Devices

The past decade has witnessed a rapid growth in the use of IMDs for monitoring and treating a variety of medical conditions, with their global market valuation expected to reach $50 billion by 2024 [17]. IMDs are increasingly being equipped with wireless interfaces [18], allowing them to communicate with an External Device (ED) such as a doctor's programmer or a patient's smartphone. While this greatly improves standard of care (allowing for post-deployment tuning of therapy as needed and remote, real-time access to health data), it also exposes the IMD to a range of security concerns [3, 4, 19–21] such as potential interaction with untrusted EDs and potential leakage of confidential medical data.

Several techniques [22–30] have been proposed to address these concerns. One such technique [28, 30] leverages a *trusted entity* such as a Health Server (HS), which assumes the role of an authenticator and arbitrates access to an IMD when an ED requests it. However, there are three unaddressed challenges with this approach. First, due to their severely limited energy budget, IMDs [18] typically utilize short-range communication technologies such as Bluetooth LE and, hence, do not have direct network connectivity with a remote HS. This is in contrast to an ED, such as a smartphone, which can use long-range wireless technologies such as Wi-Fi, LTE, *etc.*, for network access. Second, involving the HS for every single authentication session can incur significant energy overheads at the IMD, besides increasing the load on the HS and the network. This, in turn, gives rise to the third challenge. If the HS is not reachable over the network, the IMD and ED cannot securely communicate with each other. Besides, some of these techniques [28, 29] require the (resource-constrained) IMD to perform asymmetric cryptography operations resulting in high energy overhead.

Chapter 5 presents the design of a lightweight end-to-end authentication and key-exchange protocol [10] based on the trusted-entity approach that fully addresses each of the aforementioned challenges. A key requirement of the proposed protocol is the availability/generation of unique identifiers/keys and random numbers on demand. Although other techniques such as statically-stored secret keys and pseudo-random number generators could be utilized, we utilize a PUF to satisfy this requirement and present its seamless integration with existing cryptography techniques in the protocol. The result is a robust, secure, and lightweight protocol, which protects against various security attacks [19–21] and can be easily implemented using Commercial-Off-The-Shelf (COTS) devices with minimal or no additional hardware resources. It also overcomes the shortcomings associated with prior PUF-based authentication techniques [29–32] and those of earlier trusted-entity approaches [28].

The rest of this dissertation is organized as follows. Chapter 2 serves as the necessary background for this dissertation and also discusses prior work. Chapter 3 presents the design of the proposed DRAM PUF alongside a robust, low-overhead mechanism for performing device authentication across temporal and environmental variations. Chapter 4 motivates the need for multi-component authentication in an embedded system and presents the design of the proposed memory-based combination PUF. Next, Chapter 5 highlights the need for authentication and key-exchange in an

IMD ecosystem and presents the design of the proposed lightweight authentication and key-exchange protocol. Finally, Chapter 6 concludes the dissertation.

2. BACKGROUND AND RELATED WORK

As discussed in the previous chapter, there are several challenges associated with building a secure IoT. One way of addressing several of these challenges, which the current dissertation focuses upon, is device authentication. In this dissertation, we present three research works towards enabling low-cost and low-overhead device authentication in embedded systems. Before moving into the details of their design, we discuss the necessary background and related work in this chapter.

2.1 Device Authentication

Authentication can be defined as a process by which a trusted system (authenticator) verifies the identity of an untrusted entity (client) before granting it access to any data or resources. It is usually performed using a challenge-response mechanism [8,33], as depicted in Fig. 2.1. The authenticator hosts a service that is restricted to genuine clients only. Note that, in other cases, the authenticator may not host the service itself but only arbitrate access to a server that does. To verify the identity of a client, the authenticator first provides it with a challenge. The client then generates a response to the challenge. Prior to this, the authenticator creates a Challenge-Response Pair (CRP) database (Fig. 2.1) that stores all the challenges and their expected responses from genuine clients. By comparing the current client's response against the one stored in the CRP database, the authenticator infers whether the client is genuine or not. As shown, Client-1 (genuine) passes authentication and is granted access to the service as its response (R_1) to the challenge (C) matches with the CRP database. On the contrary, the response (R_2) generated by Client-2 (fake) is different from the expected response of a genuine client, and hence Client-2 is not authenticated.

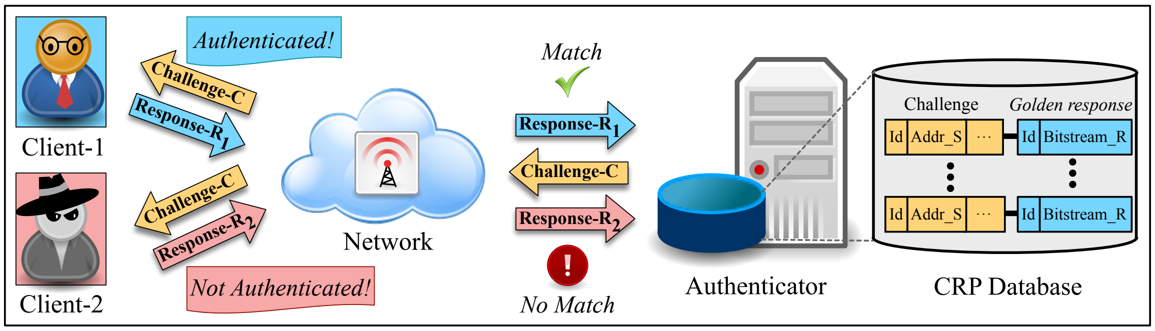


Fig. 2.1.: Overview of challenge-response-based authentication

In the context of the IoT, this challenge-response mechanism leads us to *device authentication*. The client, here, could refer to a device trying to connect to a (private) network while the authenticator is a trusted authority and could refer to the network gateway. By ensuring that each of these devices is *authentic*, in other words, by performing device authentication, a secure network can be built (Fig. 1.2).

2.2 Physically Unclonable Functions (PUFs)

Traditionally, device authentication has been carried out using static techniques such as passwords, secret keys, public-key infrastructure (PKI), *etc.* However, these techniques have several shortcoming such as the need for user intervention and vulnerability to present-day sophisticated attacks. Most importantly, a large number of the IoT devices are low-cost in nature and lack the resources – compute, memory, energy, *etc.,* which are required by some of these authentication techniques. All of these motivate us to look into alternate techniques for performing device authentication. One such technique is based on the use of hardware-intrinsic security mechanisms such as Physically Unclonable Functions (PUFs) [6]. A PUF maps a set of challenges to a set of responses based on random physical variations during the manufacturing of a device (containing the PUF). As a result, the challenge-response behavior of the PUF is highly unpredictable. Also, the fact that it is impossible to manufacture a PUF with the same behavior as another makes it unclonable and unique. These features make PUF an ideal candidate for authentication and random number generation. Randomly generated secret keys are used by various cryptographic applications such as keyed-hash message authentication code (HMAC), encryption/decryption, *etc.,* besides serving as unique fingerprints or signatures that can be used to identify a device. A PUF enables the generation of secret keys on demand rather than permanently storing them in non-volatile memory, drastically reducing the implications of physically invasive attacks. Device authentication can be considered an extension of the above key-generation process but involves the challenge-response mechanism, described earlier, to authenticate the device (client). The authenticator sends the challenge to the device, which utilizes the PUF present inside it to respond to the challenge.

One of the earliest works on PUFs was carried out in [34], resulting in an optical PUF based on the scatter pattern of a laser beam. Ref. [6] introduced the concept of silicon PUFs and provided various circuit realizations that could be integrated into an electronic circuit. Due to this ease of integration, silicon PUFs have become extremely popular in present day implementations. Two such examples are Ring Oscillator and Arbiter PUFs [33] that exploit the inherent delay characteristics in IC components for authentication and generation of secret keys. However, both these implementations require dedicated circuitry that is added solely for the PUF operation and present an area overhead. Memory-based PUFs, which form another type of silicon PUFs and are utilized in this dissertation, are discussed next. Ref. [35] proposed a generic PUF architecture targeted towards preventing modeling attacks.

2.2.1 Memory-based PUFs

Memory-based PUFs utilize the memory module (SRAM, DRAM, *etc.*) already present on the IC/SoC and require minimal (or no) additional circuitry for their operation, giving them a distinct advantage over other PUF implementations. Hence, this dissertation utilizes two such PUFs – DRAM PUF and SRAM PUF, based on the *refresh-pausing* and *power-cycling* approaches, respectively. These are explained next.

2.2.2 DRAM: Structure and PUF mechanisms

Fig. 2.2 shows the fundamental building blocks of a DRAM bit cell, namely an access transistor (M) and capacitor (C). The bit-value is decided by the charge on the capacitor; full charge implies ‘1’ and no charge implies ‘0’, or vice-versa. This charge leaks over time due to several factors related to the non-ideality of the access transistor and eventually results in the loss of data stored in the cell. This phenomenon is referred to as a *bit-flip* (‘1’→‘0’ or ‘0’→‘1’) in DRAMs. To prevent this, the DRAM memory-controller refreshes the cells (replenishes the charge) periodically (*e.g.,* every 64ms). Due to process variations, the rate of leakage (or bit-flip) varies widely across DRAMs (and within the same DRAM). This forms the basis of the *refresh-pausing* approach in a DRAM PUF [7, 8, 11, 36, 37], in which refresh operations are (intentionally) paused for a certain time-interval, generating unique bit-flip patterns in the DRAM data. This data is then read out and forms the PUF’s response. Unlike the SRAM PUF (described above), the parameters in a DRAM PUF’s challenge can be extensively varied [8], supporting an exponential number of CRPs (Section 4.3.1).

Several DRAM PUFs have been proposed by researchers over the years. Reduction of the write-duty cycle in a DRAM module is employed in [12] to implement a strong PUF. However, the reduction is achieved by adding a delay generator to the write-circuitry of the DRAM module. This not only requires very precise control over the *write* signal but is also not applicable to off-the-shelf DRAM modules. Ref. [11] utilizes refresh pausing to generate unique identifiers and random numbers from a DRAM module. However, it employs a constant (and large) refresh-pause interval of 256 s - 8192 s that could be considered too slow for authentication. A decay-based (or refresh-pausing-based) DRAM PUF that can be accessed during run-time in a commodity device is presented in [38]. However, just like [11], the lack of DRAM characterization

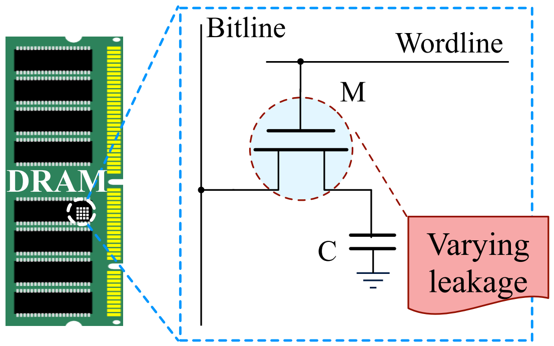


Fig. 2.2.: A 1T-1C DRAM bit-cell

in the proposed design leads to the usage of large refresh-pause intervals (120 s - 320 s) and hence, leads to slow authentication. Ref. [39] generates random numbers from DRAM using remanence effects but requires power-cycling (power off → power on → read data) of the DRAM module. The power-cycling approach has also been utilized to generate the fingerprint from the start-up values of a DRAM module [13, 40]. Modifying read access latency to generate error patterns formed the basis of the DRAM PUF presented in [41]. Other approaches to realizing DRAM PUFs [42] have also been developed. However, the *refresh-pausing*-based DRAM PUF, which this dissertation utilizes, remains as one of the most popular and widely-adopted in several state-of-the-art systems.

2.2.3 SRAM: Structure and PUF mechanisms

Each cell (or bit) in a Static Random Access Memory (SRAM) is arranged in a six-transistor configuration[1] consisting of cross-coupled CMOS inverters (M_1–M_4) and access transistors (M_5–M_6), as shown in Fig. 2.3. Powering-up the SRAM causes each cell to reach one of two states, [Q=1, $\overline{Q}$=0] or [Q=0, $\overline{Q}$=1], depending upon the relative strengths of the transistors as well as noise.

[1]While other SRAM cells do exist, the 6-transistor cell is commonly used.

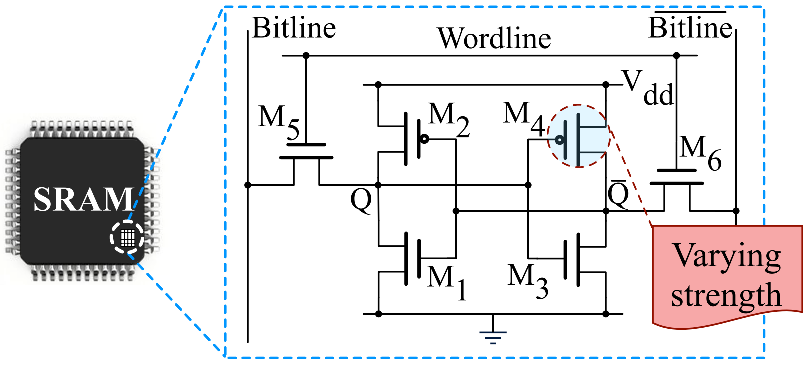


Fig. 2.3.: A 6T SRAM bit-cell

Inherent process variations during the manufacturing process cause these strengths to vary across SRAMs (and also within the same SRAM) leading to the data values in every SRAM being different immediately after start-up. This forms the foundation of an SRAM PUF [43, 44] that follows the *power-cycling* (power off → power on → read SRAM) approach to generate unique start-up values as responses. The challenge, here, specifies the address of the block inside the SRAM from where the start-up value is to be read as well as its size (*i.e.,* number of bits).

A large body of work has also focused on building SRAM PUFs. Ref. [43] extracted unique fingerprints and generated true random numbers by using the power-up state (start-up value) of an SRAM chip. Another SRAM-based PUF was presented in [45], which utilized error-patterns in caches resulting from supply-voltage reduction. This approach enables run-time generation of SRAM fingerprints since it does not require *power-cycling*. Ref. [46] observed variations in read current with the stored content in an SRAM array and used them to extract unique fingerprints. Write failures are (intentionally) introduced in an SRAM array through a programmable wordline duty-cycle controller for generating unique responses in [47]. Ref. [48], on the other hand, proposes writing values into a column of SRAM cells, followed by concurrent activation of multiple wordlines. Thus, multiple cells in the column are read at the same time, forming the PUF response. In this dissertation, we utilize the *power-*

cycling-based SRAM PUF as it remains one of the most popular and widely-adopted in several state-of-the-art systems.

2.2.4 Other Memory-based PUFs

Non-volatile memories have also been explored as potential PUF implementations. Refs. [49,50] proposed mechanisms to extract device fingerprints from flash memories; the latter also proposed true random number generation using a similar approach. Ref. [51] proposed a scheme that utilizes NVM (*e.g.,* memristor, flash, *etc.*) cells to realize a PUF without using any *helper data* (for error correction).

2.2.5 Reconfigurable PUFs

A parallel approach to the development of strong PUFs has focused on reconfiguration. Reconfigurable PUFs (rPUFs) [52, 53] have a mechanism to transform themselves, generating a new and unpredictable challenge-response behavior. Ref. [54] implements a *logically* reconfigurable SRAM PUF for secure key storage by hashing the start-up state of the SRAM with a stored bitstream, referred to as the logical *state*.

2.2.6 Strong and Weak PUFs

PUFs have been divided into two broad categories [55] - strong PUFs and weak PUFs. Strong PUFs [7, 8, 11, 38, 46, 48] can support a very large number of CRPs and are well suited for authentication. On the other hand, Weak PUFs [43, 44] are primarily used for secret key generation as they support a relatively much smaller number of CRPs.

2.2.7 Combination PUFs

Combining (the output of) multiple locations of one or several memory blocks located inside an IC to derive unique keys was mentioned in a patent [15]. By using an address decoder that can permute access order across different memory locations/blocks in a potentially unknown manner, the work claims to increase the resistance of the PUF against invasive attacks. However, no physical implementation or test results thereof have been published yet. A similar but non-memory-based PUF was presented in [16], which proposes combining several on-chip entropy sources (*e.g.,* clock sinks) in an optimized manner towards better (overall) entropy and robustness.

2.3 Authentication in Implantable Medical Devices (IMDs)

An increasing number of Implantable Medical Devices (IMDs) are equipped with wireless interfaces [18], allowing them to communicate with an external devices such as a doctor's programmer or a patient's smartphone, *etc.* While this greatly improves standard of care, it also exposes the IMD to a range of security concerns [3, 4, 19–21] such as potential interaction with untrusted devices and potential leakage of confidential medical data. Several techniques [22–30] have been proposed to address these concerns. We discuss some of these next.

2.3.1 Trusted-entity Approach and the IMD Ecosystem

One technique of addressing the above-mentioned concerns with IMDs [28, 30] leverages a *trusted entity* such as a Health Server (HS), which assumes the role of an authenticator. Fig. 2.4 shows an IMD ecosystem based on this trusted entity and comprises of three (types of) entities: Implantable Medical Device (IMD), External Device (ED), and Health Server (HS). IMD is a small device that is embedded in a patient to monitor or treat a certain physiological condition. Typical examples of

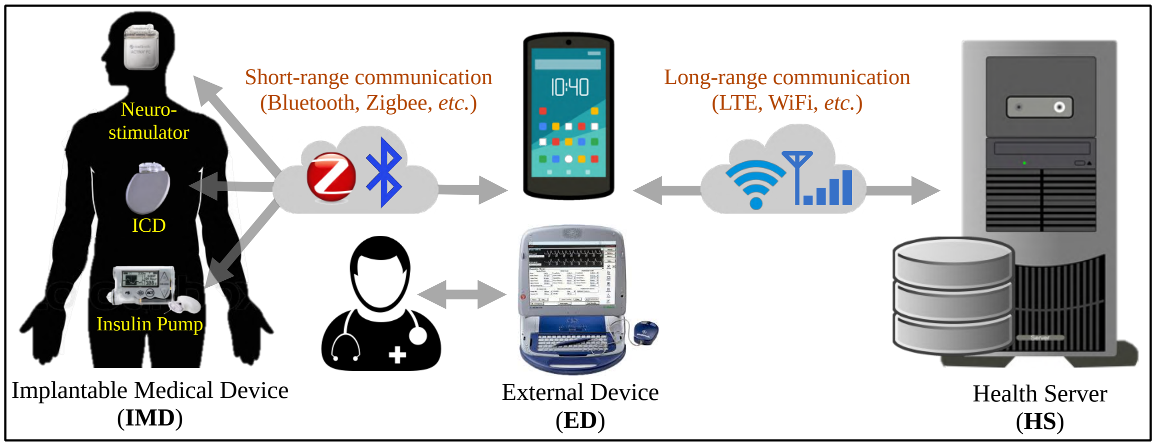


Fig. 2.4.: An End-to-End IMD Ecosystem

IMD include Implantable Cardioverter Defibrillator (ICD), insulin pump, and neurostimulator. ED, on the other hand, is an external device that is used to program or configure IMD (after surgical implantation) as well as process the physiological data relayed by it for monitoring and therapy. Hence, ED is also known as a *programmer* in some literature and is typically a hand-held device carried by a doctor or medical practitioner. Note that such a device can also be carried by the patient for day-to-day self-monitoring. The third entity in this ecosystem is HS, which serves as the trusted entity itself. A typical example of HS is a patient database server maintained by a healthcare provider. The HS arbitrates access to an IMD when the ED requests it. Upon successful authentication of the ED, the HS distributes a shared key to both the IMD and the ED. This key is used to encrypt all further communication between the IMD and ED. Note that the trusted-entity approach is popular due to its simplified but secure usage model and requirement of no additional devices.

2.3.2 Other Approaches to IMD Authentication

Over the years, several other approaches have also been proposed to enable authentication and key exchange in IMDs. Proximity-based approaches [24] authenticate an untrusted entity (*e.g.,* ED) based on its proximity (or distance) from the authenticating entity (*e.g.,* IMD). Biometric approaches [25, 26], on the other hand, adopt a *touch-to-access* policy, *i.e.,* they use a patient's biometric or physiological values for authentication and extraction of shared keys. Another technique to authentication and key-exchange in IMDs uses a side-channel such as vibration [27]. A large body of work is proxy-based, *i.e.,* use an external (patient-carried) proxy device [26], which is trusted and mediates communication with the IMD. Lastly, the trusted-entity approach, described earlier, is used in several works [28, 30] to arbitrate access to the IMD and distribute shared keys. While each of the aforementioned approaches has its advantages and disadvantages [20,22], we adopt the trusted-entity approach in the current work due to its better security capabilities, simplified usage model (for the patient), and the potential for seamless integration with Commercial-Off-The-Shelf (COTS) devices.

In recent years, PUFs have been used to implement several IMD-specific [29,30] as well as generic (not IMD-specific) authentication protocols [31,32]. Ref. [29] proposed utilizing two PUFs, one in an intra-body IC (IMD) and one on an FPGA (ED), both of which are *matched* to produce the same response. However, the proposed technique forces the use of an FPGA and requires a cumbersome pre-deployment *matching* process. Also, it uses asymmetric (public-key) cryptography, which is significantly more expensive than symmetric key cryptography in terms of energy consumption. Ref. [30] utilizes PUF-generated physically-obfuscated (or secret) keys stored in two IC cards, belonging to the doctor (ED) and patient (IMD) respectively, for authentication and key-exchange between the two entities. However, the protocol requires the services of HS during every authentication session, which has several disadvantages such as high overhead at the IMD and HS as well as service disruption in the event of HS' unavailability. Additionally, both the aforementioned works need special hardware, *viz.* an FPGA and an ASIC, and are not amenable to implementation on COTS devices. Ref. [31] presents a survey of several generic authentication and key-exchange protocols. These protocols are designed for a two-entity ecosystem, where the entities can communicate directly with and authenticate each other. On the other hand, as shown in Fig. 2.4, the IMD ecosystem in the current work comprises of three entities, two of which (IMD and HS) do not have any direct connectivity with each other. As a result, these protocols are not directly applicable to the current ecosystem. Another generic PUF-based protocol was described in Ref. [32], which performs authentication and key-exchange between a resource-constrained prover (containing a PUF) and a resource-rich verifier without requiring the PUF's (prover's) challenge-response pairs to be stored at the verifier end. Moreover, it assumes that the verifier has direct connectivity with the trusted-entity. On the contrary, the current ecosystem is based on IMD (verifier) being resource-constrained while ED (prover) being resource-rich. IMD (verifier) also does not have direct connectivity with HS (trusted-entity), and hence, the protocol described in Ref. [32] is not applicable to the current ecosystem.

3. AN INTRINSICALLY RECONFIGURABLE DRAM PUF FOR DEVICE AUTHENTICATION AND RANDOM NUMBER GENERATION

Over the past few years, researchers have taken an active interest in hardware-intrinsic security mechanisms, and in particular, Physically Unclonable Functions (PUFs) [6]. PUFs have proved to be a secure, low-cost, and robust authentication measure against issues of counterfeiting and information leakage. They exploit the random physical variations inherent to any manufacturing process to generate device-specific and unclonable fingerprints. These unique fingerprints can be used as the basis for a challenge-response mechanism for device authentication as well as for random number (*e.g.,* secure key) generation.

While a large variety of PUF implementations have been proposed over the years, memory-based PUFs [11, 13, 43], in particular, are an attractive candidate due to the ubiquitous presence of memory in virtually every embedded system. Moreover, memory-based PUFs require minimal or no additional circuitry for their operation, giving them a distinct advantage over other PUF implementations. One such example of a memory-based PUF, which we focus on in this chapter, is based on Dynamic Random Access Memory (DRAM). DRAM is used as the main memory in a large number of modern embedded systems (as illustrated in Fig. 3.1) due to its high density and low cost. As is well-known, DRAM cells must be refreshed periodically to preserve the stored data. This need for refresh operations also makes DRAM an attractive candidate for use as a PUF, especially for challenge-response-based authentication and random number generation.

Fig. 3.1.: Examples of embedded systems containing DRAM

3.1 Chapter Contributions

Prior approaches to DRAM-based PUFs suffer from several shortcomings such as low speed of operation [11], non-applicability to commercial off-the-shelf (COTS) devices [12], and the need for power cycling the DRAM module [13, 39]. Moreover, the near static nature of the response-generation mechanism in some of the works [11, 13] makes them vulnerable to various security attacks [14]. To address these limitations, we propose a DRAM PUF based on refresh pausing that not only supports a very large number of challenge-response pairs (CRPs) through the variation of different parameters but is also *intrinsically reconfigurable*, *i.e.*, its challenge-response behavior can be substantially modified without the use of any additional circuitry. Hence, the PUF can be easily implemented in most off-the-shelf systems and provides considerable protection from various security attacks. Specifically, we make the following contributions [7, 8]:

- We perform a comprehensive error characterization of DRAM modules by varying different parameters (refresh-pause interval, data patterns, and temperature) to gain a deep insight into DRAM behavior. This insight allows us to systematically select DRAM blocks that are best suited for use in a PUF.

- We propose an *intrinsically reconfigurable* DRAM PUF (D-PUF), based on refresh pausing, for device authentication and random number generation. Reconfiguration is achieved through variation of the refresh-pause interval, altering the challenge-response behavior of the PUF and making it robust against various attacks. We also use D-PUF to design a secure, low-overhead mechanism for performing device authentication. The mechanism operates robustly even in the presence of environmental and temporal variations.

- We implement D-PUF and our proposed authentication mechanism in a real system using off-the-shelf DRAM modules and evaluate it thoroughly. In particular, we demonstrate a 4.3X-6.4X reduction in authentication time, compared to previous work. Using controlled temperature and accelerated aging tests, we demonstrate the robustness of our authentication mechanism to temperature and aging effects. For a sample temperature range of 10°C to 60°C, we show that the mechanism achieves a 100% true-positive (successful authentication) rate and 0% false-positive rate. For a ten-month-old DRAM module, it also ensures a 100% true-positive rate and 0% false-positive rate.

- We implement a True Random Number Generator (TRNG) using D-PUF by exploiting the Variable Retention Time (VRT) phenomenon in DRAMs and demonstrate that the generated random numbers pass all the tests specified in the NIST Statistical Test Suite.

The remainder of this chapter is organized as follows. Section 3.2 describes the motivation behind this work. Next, Section 3.3 presents the D-PUF design while Section 3.4 describes the mechanisms for authentication and random number generation using D-PUF. Section 3.6 and Section 3.7 describe the experimental setup and present the results of our experiments, respectively. A discussion about implementation-related considerations is provided in Section 3.8. Finally, Section 3.9 concludes the chapter.

3.2 Motivation

Memory-based PUFs have a distinct advantage over other PUF implementations as they use components (SRAM, DRAM, *etc.*) that are inherent to most modern embedded systems. Hence, they require minimal or no additional circuitry for their operation and could enable energy-efficient designs for emerging IoT devices [56]. Over the past few years, SRAM-based PUFs have been widely studied and used as security primitives for various state-of-the-art systems [43, 44, 47, 54]. However, these PUFs suffer from several shortcomings such as limited entropy, the requirement of power cycling, high cost, *etc.*, and are therefore limited to applications that require a very few number of CRPs or random numbers. DRAM-based PUFs, with their large address space and high density, have the potential to support a large number of CRPs (or random numbers). However, current designs suffer from several shortcomings such as the requirement of power cycling [13, 40] and large refresh-pause intervals [11, 38]. Moreover, weak (memory-based) PUFs are vulnerable to various sophisticated attacks [14]. One way of guarding against such attacks is altering the challenge-response behavior of a PUF, in other words, *reconfiguring* the PUF [52–54]. However, these designs require additional hardware resource (*e.g.,* private NVM) to achieve reconfiguration, and hence, cannot be applied to commercially available DRAM module.

3.3 D-PUF: An Intrinsically-reconfigurable DRAM PUF

Towards addressing the shortcomings of memory-based PUFs described earlier, we propose an *intrinsically reconfigurable* DRAM PUF (D-PUF) based on refresh pausing. As described in Chapter 2, refresh pausing is a lightweight approach towards implementing DRAM PUFs and involves stopping the refresh operations in a DRAM module for a specified interval. In DRAMs, inter-die and intra-die variations lead to highly variable bit-cell strengths distributed randomly across different modules as well as within a module. During refresh pausing, these variations cause DRAM cells to leak charge at different rates, resulting in highly random yet unique bit-errors in the data stored in the DRAM module. We exploit this randomness (entropy) to generate CRPs (for authentication) as well as random numbers from a DRAM module. Moreover, the existence of both nature of bit-flips, *i.e.*, '1' $\rightarrow$ '0' bit-flips (true-cells) and '0' $\rightarrow$ '1' bit-flips (anti-cells) [57], in the same module enables us to extract more entropy out of the PUF. D-PUF is also designed to be *intrinsically reconfigurable* – the choice of refresh-pause interval as the reconfiguration parameter (described later) enables alteration of the PUF behavior without the requirement of any additional resource. Hence, it can be easily applied to commercially available DRAM modules in contrast to [12, 54]. It also alleviates the problem of a large refresh-pause interval, which was required in [11, 38], and also performs robust authentication across a comparatively much wider operating range (temperature and aging). The refresh pausing approach also ensures that there is no need for power cycling, unlike [13, 39].

In this work, we try to realize two important functions using a DRAM PUF - *(i) authentication* and *(ii) random number generation.* As described in Chapter 2, *authentication* refers to any process by which a trusted entity verifies the identity of another entity trying to access the former's services. We propose a secure, low-overhead mechanism that uses D-PUF and performs device authentication without the need for additional cryptographic resources, which are used in traditional encrypted-password-based methods. The authentication is also assisted by a low-complexity algorithm for selecting the DRAM blocks in D-PUF that ensure the minimum required entropy at the lowest refresh-pause intervals. *Random number generation* involves producing a sequence of numbers that cannot be predicted better than by random chance. Random numbers find uses in various cryptographic applications such as HMAC and encryption/decryption as well as in Monte Carlo simulations, statistical research, *etc.* These functions require D-PUF to generate a response and random number, respectively; we briefly describe the sequence of steps involved in these processes here. A random binary bitstream of a particular size, referred to as *challenge*, is first written onto a specified memory address in D-PUF, following which, the memory controller pauses the refresh operation for a pre-decided time interval, called the *refresh-pause interval*. Next, the binary bitstream is read out from the same memory address and processed for subsequent error correction using some information (helper data) already stored at the device containing the PUF. The error-corrected bitstream is then sent out as the *response*. For generating a random number, the read-out raw bitstream (without error correction) is applied to a suitable hash algorithm (*e.g.,* SHA-256), to produce a random binary sequence. Whenever needed, reconfiguration is carried out by simply changing the refresh-pause interval associated with the generation of the response or random number.

Next, we describe device authentication and random number generation using D-PUF in detail.

3.4 Device Authentication using D-PUF

The device authentication mechanism in D-PUF is divided into three phases - *(i) characterization* phase, *(ii) enrollment* phase, and *(iii) authentication* phase. Fig. 3.5, 3.6, and 3.7 present flow diagrams of these different phases, which are explained later in detail. The dotted lines between the device (containing the PUF) and the authenticator represent the interactions between them, while the solid lines show the actions inside each of these entities. The *characterization* phase ensures the usage of the minimum refresh-pause interval for authentication by carrying out a coarse-grained error characterization of the DRAM module (used as the PUF) and simultaneously guarantees that it meets the minimum required entropy. During *enrollment* phase, the PUF responds to random challenges sent by an authenticator, generating a CRP database that is stored at the latter. It is followed by the *authentication* phase, where responses are generated by the PUF for one or more challenges picked from the CRP database. These responses are then compared to the ones stored in the database; if there is an exact match or the difference is within a small threshold, the device containing the PUF is authenticated. Note that by varying different pa-

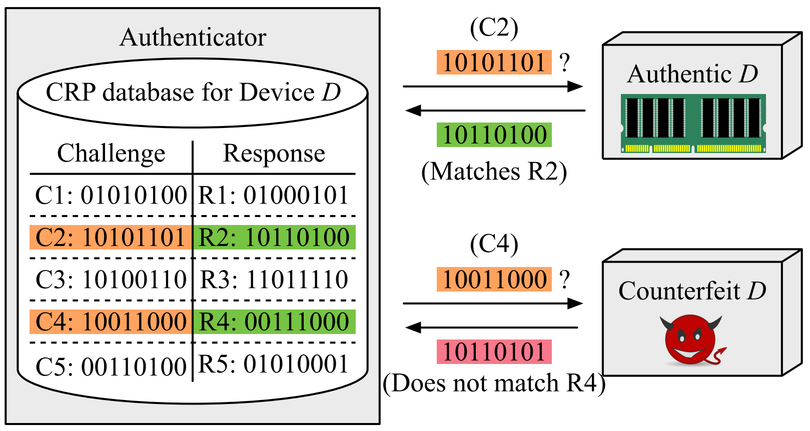


Fig. 3.2.: Authentication using D-PUF

rameters associated with the generation of a response, a very large number of CRPs can be generated for carrying out challenge-response-based authentication. Hence, D-PUF closely resembles a strong PUF. Fig. 3.2 provides a high-level overview of the entire process.

Before delving into the details of the authentication mechanism, we define a few terms and briefly discuss our assumptions. We also present the formats of the challenge and response used in our design below.

3.4.1 Definitions and Assumptions

- *Device* (D): An untrusted client device that requests authentication and contains D-PUF. It is assumed to possess sufficient computational and memory resources to perform error correction and store kilobytes of binary data. An example of such a device is a smartphone.

- *Reconfigurable DRAM PUF or D-PUF* (P): The DRAM module that implements PUF functionality in the device D. When we mention that a response is generated by D, it should be assumed that the same is actually generated by D-PUF (P) present inside D.

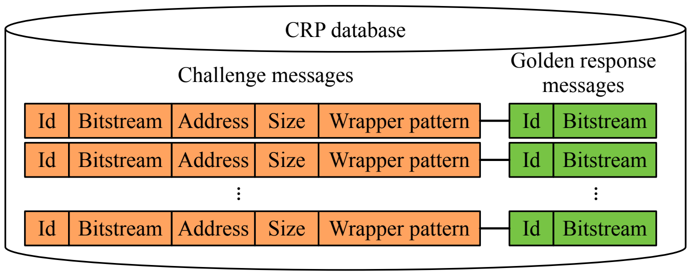


Fig. 3.3.: Challenge and Response Format and CRP database

- *Authenticator* (A): A trusted party which authenticates the device D. A is assumed to have access to the CRP database and limited information about the characteristics of P. It possesses greater computational and memory resources than D to process and store gigabytes of data, *e.g.,* a server.

3.4.2 Challenge and Response Message Format

In the proposed design, challenges and responses are represented as 5-tuple and 2-tuple messages respectively, as depicted in Fig. 3.3. An entry in the CRP database comprises of a *challenge* message (CM) and a *golden response* message (GRM) (explained later in Section 3.4.4).

The response generated during authentication is referred to as a *response* message (RM). *Id* refers to the index number assigned by A to an entry in the CRP database. *Bitstream* is a random binary sequence of *size* bytes generated by A (in a CM) or D (in an RM and a GRM). Specifically, in a CM, *bitstream* is the data written onto P while in an RM and a GRM, it refers to the data read from P. *Address* specifies the starting memory address in P where this write or read occurs. *Wrapper pattern* represents a predefined binary sequence and is explained next.

Wrapper pattern We observed that the number, position, and nature of bit-flips in a response generated by a DRAM block in P were influenced by the peripheral data-bits surrounding the block (Section 3.7.2). These peripheral data-bits, written just before the beginning and after the end of the block, are collectively referred to as wrapper data and are specified by a *wrapper pattern*. The *wrapper pattern* is a part of the challenge message and can be one of the several predefined types, *e.g.,* all 1s, all 0s, checkered, *etc.* The challenge *bitstream* is padded with the corresponding wrapper data before it is written into the block. However, the wrapper data is not part of the response message that is sent back to A. *Wrapper pattern*, thus, serves as another variable parameter for CRP generation.

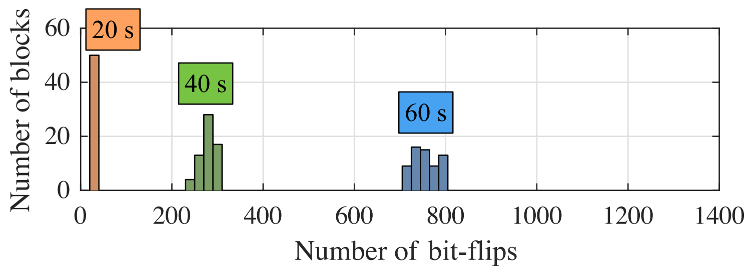
3.4.3 Characterization phase

The characterization of a DRAM module (P) involves understanding its bit-flip behavior, which is affected by numerous factors, *e.g.,* refresh-pause interval, input-data pattern, temperature, *etc.*, as well as the DRAM module itself. We utilize the characterization results to derive vital insights that enable D-PUF to have a lower authentication overhead, compared to prior art, while meeting the specified entropy requirements. Besides, they guide the design of a robust authentication mechanism across a wide range of operating conditions. A step-by-step description of the characterization methodology is given later in Section 3.6; Section 3.7.2 later shows how the bit-flip behavior is affected by various factors (parameters) in one of the DRAM modules.

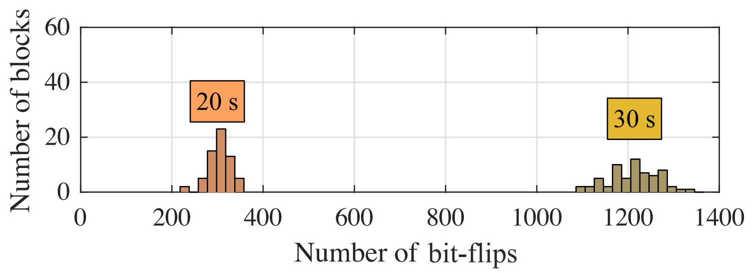
Fig. 3.4 shows the characterization results for two DRAM modules at different refresh-pause intervals with other parameters remaining the same. A few key observations from these results that are leveraged in the D-PUF design are given below:

1. A conservative (and fixed) refresh-pause interval [11] satisfies entropy requirements but can lead to slow authentication in DRAM PUFs based on refresh pausing.

2. Given a refresh-pause interval, some blocks in the DRAM address space may not contain the required entropy (bit-flips), making them unsuitable for PUF



(a) Module A



(b) Module B

Fig. 3.4.: Variation of bit-flips across different DRAM modules

operation. For example, only some of the blocks in Module A satisfy an entropy requirement of 250 bit-flips at 40 s refresh-pause interval, as shown in Fig. 3.4(a).

3. The variation in entropy is more pronounced across modules, making a constant refresh-pause interval potentially unsuitable for some modules. For example, for the same entropy requirement of 250 bit-flips, a relatively lower interval of 20 s is suitable for Module B, as shown in Fig. 3.4(b), unlike in the case of Module A.

4. Higher temperatures result in an exponential rise in the number of bit-flips, which can potentially hinder the authentication process, as described later in Section 3.7.4.

Fig. 3.5 shows the flow diagram of the proposed *characterization* phase where the numbers adjacent to each rectangle specify the sequence of operations during the phase. It starts with the generation of the characterization results (C) of a sub-address space (S) in the DRAM module, P (in D), as described in Section 3.6 [step 1].

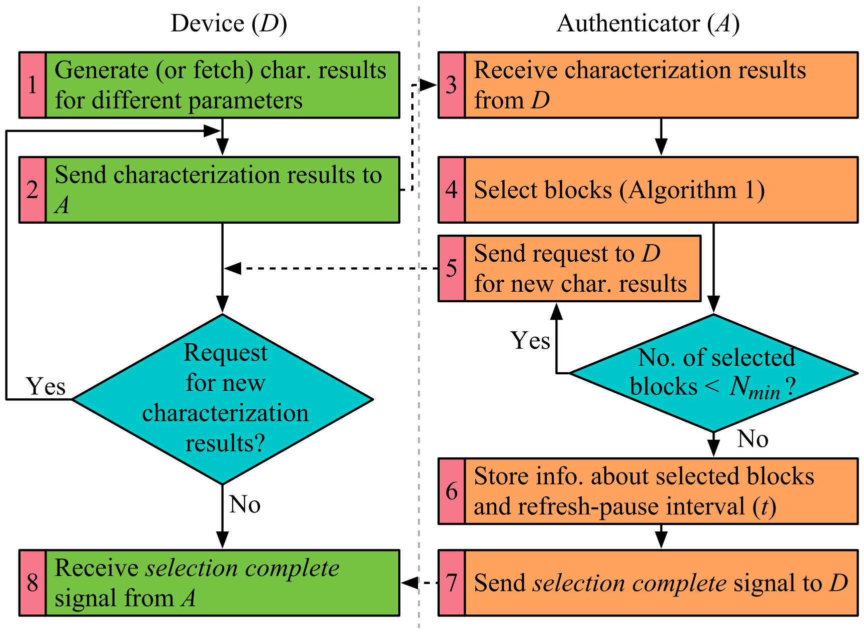

Device (D)
Authenticator (A)
1 Generate (or fetch) char. results for different parameters
2 Send characterization results to A
3 Receive characterization results from D
4 Select blocks (Algorithm 1)
5 Send request to D for new char. results
Yes
No. of selected blocks < Nmin ?
No
Request for new characterization results?
Yes
No
6 Store info. about selected blocks and refresh-pause interval (t)
7 Send selection complete signal to D
8 Receive selection complete signal from A

Fig. 3.5.: D-PUF *Characterization* phase

The characterization results provide insights into the number, position, and nature of bit-flips within P. Note that these results are generated for a certain refresh-pause interval, t. Ideally, t should be as low as possible. However, the lowest possible refresh-pause interval may have already been used for the sub-address space (S), and hence may not be available for use. D sends these results to the authenticator, A, which uses it to choose the best blocks that meet the entropy requirements at the lowest available refresh-pause interval [steps 2, 3]. Algorithm 1 describes the proposed pseudocode for the selection of blocks [step 4]. Note that a very low refresh-pause interval (t) may lead to none (or very few) of the blocks in a particular sub-address space (S) meeting the entropy requirements. As the block selection (and characterization) process is computation and time-intensive, a sufficient number of blocks ($\mathrm{N_{min}}$, decided by the PUF designer) should be selected up front to create enough CRPs without the need for re-characterizing (and re-selecting blocks) anytime soon. Hence, in case the number of selected blocks is less than $\mathrm{N_{min}}$, A may request D to provide either the characterization results for a new sub-address space (S') at

t or those for the same sub-address space (S) at a higher refresh-pause interval, t' ($t<t'$) [step 5]. Moving to a new sub-address space (S') while still utilizing the lower refresh-pause interval (t) could later enable fast authentication (during the *enrollment* phase). But, it could exhaust the total available address space in D quickly. Moving to a higher interval (t'), on the contrary, could result in relatively slower authentication but will exhaust the address space slowly. While the former approach is suitable for PUFs that have a large address space and require fast authentication, the latter is suitable for PUFs that have a small address space but can tolerate slow authentication. The PUF designer can take this decision based on the *available address space vs. required authentication speed* trade-off. Finally, upon fulfillment of the "minimum number of selected blocks" criteria, A stores the information about the selected blocks (address, size, *etc.*) and the associated refresh-pause interval [step 6] and then signals D about the completion of the block selection process [steps 7, 8]. The information stored by A is later used during the *enrollment* phase for generation of the CRP database.

Block selection Algorithm 1 describes the proposed pseudocode for the selection of blocks. Block selection at A starts by finding the bit-flip positions (represented as '1' in F_{i}) for each of the input-data patterns followed by combining them to generate all possible bit-flip positions (F). Beginning with the lowest specified block-size first, selection of blocks is carried out such that they meet the minimum entropy requirements (E_{min}). E_{b} represents the entropy of a block b and is specified by the number of '1's in F corresponding to that block. Moreover, the selected blocks are non-overlapping, *i.e.,* they do not share any memory address (and bit-flips) with each other. Though it may seem like a conservative approach, it ensures that the bit-flip positions are not shared among any two blocks, and hence selects blocks with a sufficient number of unique bit-flips.

Note that an attacker may be able to get some insights into the DRAM behavior if he/she has access to the characterization results. As a result, it needs to be

Algorithm 1: Pseudocode for block selection

```
Input: $I = \{I_1, I_2, ...., I_n\}$: Set of input data-patterns for characterization,
$C = \{C_1, C_2, ...., C_n\}$: Set of characterization results of a sub-address space,
$S$, at a refresh-pause interval, $t$, for different input data-patterns,
$Z = \{Z_1, Z_2, ...., Z_m\}$: Set of specified block-sizes in increasing order,
$E_{min}$ = Minimum required entropy specified in terms of number of bit-flips
Output: $L = \{L_1, L_2, ...., L_m\}$: Set of lists of selected blocks where $L_i$ is the
        list of blocks with size $Z_i$

1  $F = \phi, L = \{\phi\}$   // $F$ represents combined bit-flips for $S$
2  for $i = 1$ to $n$ do
3      $F_i = C_i \oplus I_i$   // Bitwise XOR
4      $F = F + F_i$   // Bitwise OR
5  for $j = 1$ to $m$ do
6      $B_j = Get\_All\_NonOverlapBlocks(S, Z_j)$;
7  for $k = 1$ to $m$ do
8      foreach $b \in B_k$ do
9          $E_b = Get\_Entropy(b, F)$
10         if $E_b > E_{min}$ then
11             $OL = Check\_Overlap(b, L)$
12             if $OL = False$ then
13                 $L_k = L_k \cup b$
14     $L = L \cup L_k$
```

ensured that the characterization results are shared with the authenticator securely. Therefore, we assume that the *characterization* phase occurs in a secure environment [54, 58, 59]. The *characterization* phase is, in fact, much less frequent than the other two phases as it occurs only before initial deployment or in case D runs out of address space (in P) for the PUF operation. As regards to the characterization results, DRAM manufacturers may choose to provide them in the DRAM module itself. These could then be directly utilized by the PUF designer, thus removing the need to characterize again and saving valuable time/energy.

3.4.4 Enrollment Phase

The *enrollment* phase primarily involves the generation of the CRP database and is also assumed to be carried out in a secure environment [54,58,59]. Fig. 3.6 presents the flow diagram for the *enrollment* phase.

The device, D, starts by sending an enrollment request to the authenticator, A [steps 1, 2]. A responds by sharing the refresh-pause interval (t, stored during the *characterization phase*) with D [steps 3, 4]. Next, A constructs the challenge messages (CMs) using the block information stored in it and sends them to D [steps 5, 6]. D responds by sending back golden response messages (GRMs) that are generated using the refresh pausing approach described earlier [steps 7-12]. The golden responses together with the corresponding challenges form entries in the CRP database [steps 13, 14] against which subsequent responses are compared during the *authentication* phase. The helper data, for error correction, is also generated during the *enrollment* phase (described later) and is stored at D [step 11]. Note that the effects of variable retention time (VRT) [57, 60, 61] during the generation of golden responses could be mitigated by acquiring multiple instances of the golden responses (for the same challenge) and taking a bit-wise intersection of the same. Section 3.5 gives a brief explanation of VRT.

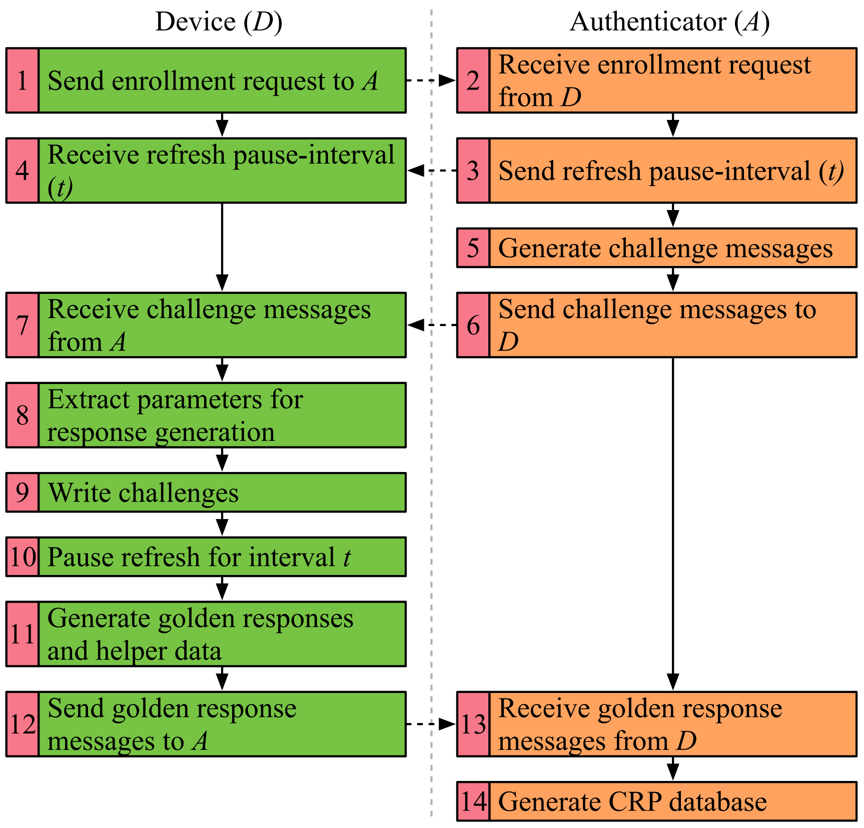

Device (D)
Authenticator (A)
1 Send enrollment request to A
2 Receive enrollment request from D
3 Send refresh pause-interval (t)
4 Receive refresh pause-interval (t)
5 Generate challenge messages
6 Send challenge messages to D
7 Receive challenge messages from A
8 Extract parameters for response generation
9 Write challenges
10 Pause refresh for interval t
11 Generate golden responses and helper data
12 Send golden response messages to A
13 Receive golden response messages from D
14 Generate CRP database

Fig. 3.6.: D-PUF *Enrollment* phase

Helper Data for Error Correction PUFs are governed by random physical processes, which are affected by numerous environmental and temporal variations. Error correction can be utilized to suppress some of these variations and enhance the reliability of the response generation process. During the *enrollment* phase, the golden responses are used to generate some redundant information that is capable of correcting the subsequent responses from D. This redundant information, known as *helper* data, is stored at D. In our work, helper data is generated by implementing a lightweight error correction algorithm - Hamming Encoder/Decoder in software.

3.4.5 Authentication Phase

Actual authentication of D is assumed to happen in an insecure environment during the *authentication* phase and is depicted in Fig. 3.7. It starts with a request for authentication from the device, D, to the authenticator, A [steps 1, 2]. A responds by selecting an entry from the stored CRP database and sending the corresponding challenge message along with the refresh-pause interval (t) to D [steps 3, 4]. The response message is then generated by D using the refresh pausing approach [steps 5-9], corrected for errors [step 10], and sent back to A [step 11]. Upon receiving the response, A calculates its hamming distance (HD) from the corresponding golden response, which is stored at the selected entry in the CRP database [steps 12-13]. This HD is then compared with the *match threshold* (described next) to determine the authentication outcome that is later conveyed to D [steps 14-15].

Match threshold The existing error-correction infrastructure in D may not be sufficient to correct all the errors in the response generated during the *authentication* phase. This is more pronounced in DRAMs due to their high entropy (bit-flips) and susceptibility to several environmental and temporal variations. In such scenarios, an exact match of the generated response with the golden response (stored in the CRP database) may not happen even if D is authentic. Hence, we follow a fuzzy authentication strategy and define a *match threshold* (MT), which represents the maximum HD between the *GRM bitstream* and *RM bitstream* beyond which D is not authenticated. Refs. [43, 58] use a similar technique for unique identification of devices.

To set the appropriate value of MT, we refer to Fig. 3.16 that shows a probability distribution of the HD for five different DRAM modules at three temperatures and 50 different *CM*s. MT is given as $MT = \mu + \sigma + \tau$, where μ and σ represent the *mean* and *standard deviation* of the distribution (*same-module* comparisons). We include τ to accommodate for environmental (temperature) and temporal (aging, variable retention time, *etc.*) variations. This enables us to perform robust authentication

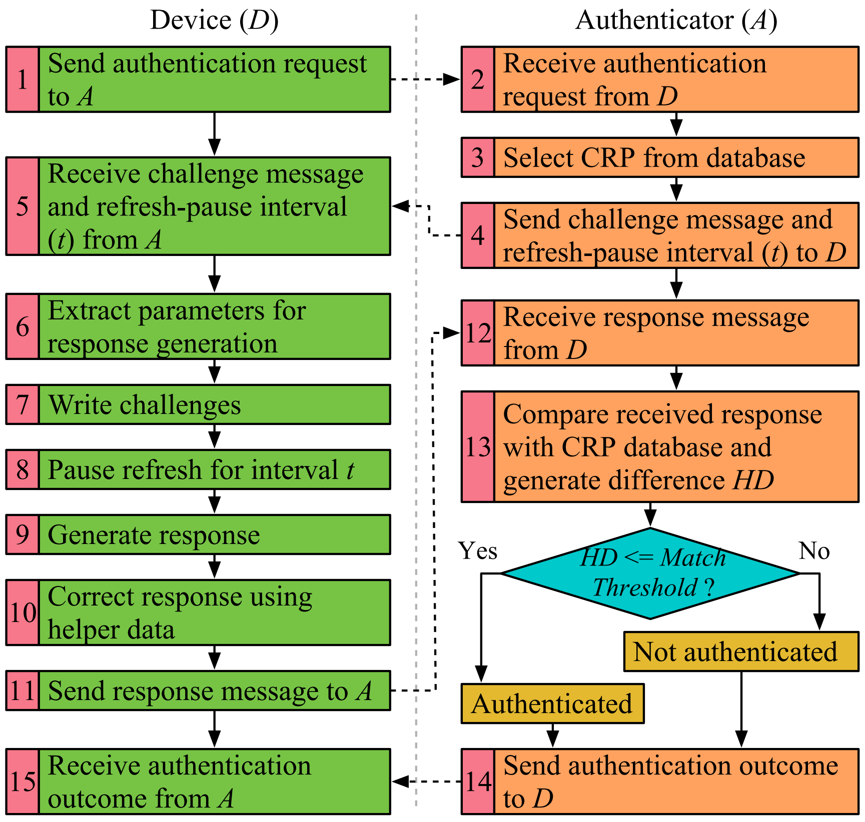

Device (*D*)
Authenticator (*A*)
1 Send authentication request to *A*
2 Receive authentication request from *D*
3 Select CRP from database
4 Send challenge message and refresh-pause interval (*t*) to *D*
5 Receive challenge message and refresh-pause interval (*t*) from *A*
6 Extract parameters for response generation
7 Write challenges
8 Pause refresh for interval *t*
9 Generate response
10 Correct response using helper data
11 Send response message to *A*
12 Receive response message from *D*
13 Compare received response with CRP database and generate difference *HD*
HD <= Match Threshold ?
Yes
No
Not authenticated
Authenticated
14 Send authentication outcome to *D*
15 Receive authentication outcome from *A*

Fig. 3.7.: D-PUF *Authentication* phase

even under high variations. Section 3.7.4 later explains all of this in detail. Note that setting the match threshold can be done by the authenticator during the *characterization* phase on the basis of the characterization results, and thus involves a one-time overhead only.

3.5 Random Number Generation using D-PUF

The overall mechanism for random number generation is divided into two phases - *(i) characterization* phase and *(ii) generation* phase. *Characterization* phase is similar to the one used in device authentication, described earlier. However, it is more rigorous in the sense that characterization runs (Section 3.6.2) are carried out with the same input data-patterns and refresh-pause intervals multiple times to identify non-deterministic (temporally random) bit-flips arising due to VRT (explained in the next sub-section) in a DRAM module. Note that this is in contrast to authentication that exploits deterministic (spatially random) bit-flips. To avoid confusion, we refer to the non-deterministic bit-flips as *random bit-flips* and the deterministic bit-flips as just *bit-flips*. As in authentication, *characterization* results are used to select blocks that meet minimum entropy requirements and are suitable for random number generation. However, entropy here refers to the number of bits that exhibit *random bit-flip*. In this context, we also define *flipProb*, a parameter that is specified by the designer and refers to the required probability with which a bit needs to flip in order to be classified as one exhibiting *random bit-flip*. For example, across 100 runs (with same input data-pattern and refresh-pause interval) and *flipProb*= 0.4, a bit is said to exhibit *random bit-flip* if it flips at least 40 times with respect to the input data-bit-value. Thus, *flipProb* helps to bring in some amount of guarantee that the selected bits shall exhibit *random bit-flips* in future also. Finally, during the *generation* phase, refresh pausing approach is used to produce the actual random number (R). However, in contrast to authentication, there is no error correction and the read-out bitstream is directly applied to a hash algorithm recursively to generate R of required size (Z). Note that hashing is utilized here to improve the randomness of the generated number. However, the output size of the hash algorithm could be lower than Z. To address this, one could perform the above described process (refresh pausing followed by hashing) multiple times and concatenate the outputs to produce the final random number of size Z. The latency of this approach could be very high if the same (one) block is only available for use every time. A better approach is presented in Fig. 3.8. As shown, if the size requirements are not met after first-time hashing, then the output is hashed again and accumulated in R. This process is repeated till the size of R is greater than or equal to Z. As hashing is much faster than performing refresh pausing, this approach helps to reduce the latency of the generation mechanism.

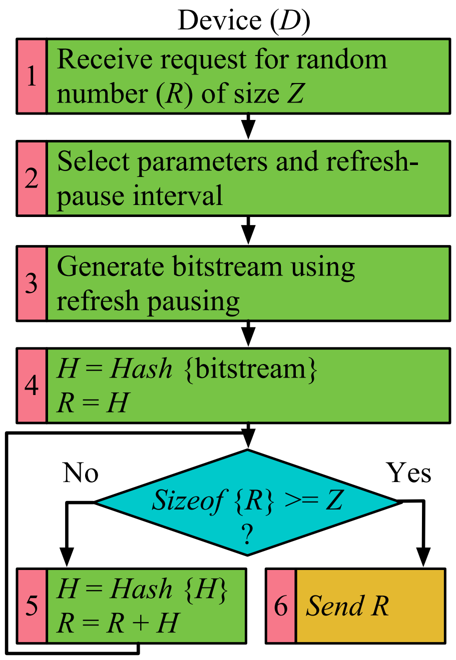

Device (D)
1 Receive request for random number (R) of size Z
2 Select parameters and refresh-pause interval
3 Generate bitstream using refresh pausing
4 $H = Hash$ {bitstream} $R = H$
No
Yes
$Sizeof\ \{R\} >= Z$?
5 $H = Hash\ \{H\}$ $R = R + H$
6 $Send\ R$

Fig. 3.8.: Random number generation using D-PUF

Alternatively, if there are enough bits in the block that exhibit *random bit-flips*, the read-out bitstream (after refresh pausing) can be directly used as the random number. Without loss of generality, we follow the former method in this work.

Variable retention time Variable retention time (VRT) is a DRAM-specific phenomenon that leads to dynamic variation in the retention-time profile of DRAM bit-cells. VRT causes DRAM cells to randomly switch between a high retention state (corresponding to high retention time) and a low retention state (corresponding to low retention time) at different points in time, as shown in Fig. 3.9. As a result, the stream of *random bit-flips*, produced due to VRT, at different time instants can act as a source of true random numbers. In the proposed work, we leverage this VRT phenomenon in DRAM cells to form the basis for random number generation. Note that VRT is present only in a fraction of the DRAM cells (called as VRT cells), which are identified by characterizing the DRAM module (P). We now briefly describe the cause behind the existence of VRT.

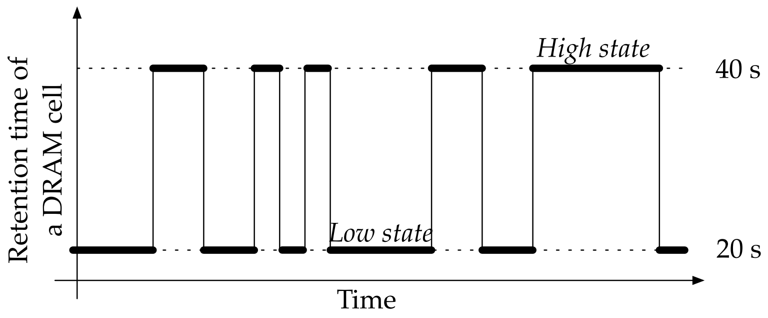


Fig. 3.9.: Different retention states in a DRAM cell due to VRT

VRT in DRAMs was first demonstrated almost three decades back [62]. The presence of traps near the gate region causes fluctuations in the gate induced drain leakage (GIDL) current in DRAM cells. The leakage current, in other words, depends on the extent to which these traps are occupied; a cell leaks faster if the traps are occupied and slower if the traps are empty. The former case leads to lower retention times while the latter leads to higher retention times [63, 64]. More importantly, the trap occupation is random in nature and varies across time. Hence, VRT cells exhibit different retention times at different instants. External factors such as high temperature during the packaging process and mechanical/electrical stress have also been shown to contribute to the VRT phenomenon.

3.6 Experimental Methodology

This section provides a brief description of the experimental methodology utilized to validate the D-PUF design.

3.6.1 Experimental Setup

All experiments were performed using an Altera Stratix IV GX FPGA-based Terasic TR4-230 development board [65], consisting of a 1GB SODIMM DDR3 DRAM. The entire experimental setup is shown in Fig. 3.10.

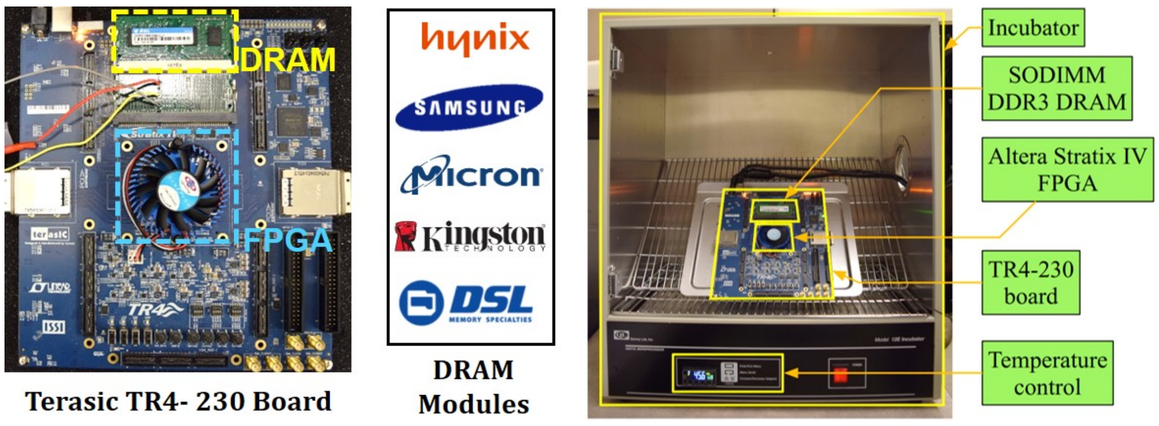


Fig. 3.10.: Experimental setup

The FPGA was programmed with a soft Nios II processor [66] along with an Altera UniPHY DDR3 memory controller for controlling the DRAM module. This controller provides the control signals required to pause the refresh operations. In modern embedded systems, these refresh control signals are usually exposed to the lower layers of the operating system. A custom slave running on the processor was also created, which can instruct the memory controller to start and stop the refresh operations. A total of six COTS 1GB DRAM DDR3 SODIMMs belonging to five different manufacturers were used for the experiments. The temperature and aging experiments were carried out by operating the DRAM modules inside the Quincy Lab 12-140E Incubator. Note that during validation, error-correction was performed using software running on the Nios II processor, while the authentication was done on a local computer connected directly to the FPGA.

3.6.2 Characterization Methodology

We now describe the details of the DRAM error-characterization process, which was performed in a number of sequential steps:

1. First, an input-data pattern was written throughout the selected DRAM sub-address space. Subsequently, the DRAM was refreshed normally (at 64 ms) so that 100% data is retained.

2. Next, the custom slave (implemented on the FPGA) disabled the refresh operations and waited for the specified refresh-pause interval, which was maintained by a precise timer controlled directly by the FPGA hardware.

3. The data from the DRAM was then read out, and normal refresh operations were restored. The acquired data was compared with the input-data to determine the number, position, and nature of bit-flips.

4. This process was repeated for different data patterns (*e.g.,* all '1's, all '0's, checkered pattern (alternate '0's and '1's), *etc.*) as well as different refresh-pause intervals (20 s, 30 s, 40 s, 60 s, 80 s, and 90 s) and temperatures (20°C, 30°C, and 40°C).

In a real system, we envision that a fixed segment of the DRAM (say 5%) will be dedicated for PUF functionality so that the DRAM can be shared simultaneously with other tasks running on the device. For generic DDR DRAMs, this segment can be selected either randomly or by an initial lightweight characterization process, where the PUF section will be refreshed at a much higher refresh interval than the rest of the module. The Partial Array Self-Refresh functionality [67] in LPDDRs inherently supports this feature and can refresh a portion of the DRAM at a different interval than the standard 64 ms.

3.7 Experimental Results

This section presents the results obtained from experiments conducted to validate our work. The results relevant to device authentication are presented first followed by those for random number generation.

3.7.1 Device Authentication

This section broadly consists of four subsections. First, we show the effects of variation of different parameters (refresh-pause interval, block-size, and wrapper pattern) on the number of bit-flips in one of the DRAM modules. An analysis of the results obtained provides useful insights for setting design parameters associated with the proposed authentication mechanism. Second, a uniqueness analysis of the responses obtained from different DRAM modules as well as from different blocks belonging to the same DRAM module is presented that shows the effectiveness of D-PUF for authentication. Third, we analyze the robustness of our proposed authentication mechanism under temperature variations using five DRAM modules. This subsection is further divided into two parts. In the first part, we present the authentication results from the DRAM modules using a naive approach that utilizes a static match threshold (MT) technique with inadequate error correction. It motivates the adoption of a dynamic approach towards setting MT and the need for adequate error correction, both of which are presented in the second part. Results show that the latter approach is much more effective under temperature variations and substantially increases the operating range - 10°C to 60°C. The last subsection analyzes the robustness of the authentication methodology under aging effects.

3.7.2 Effects of Parameter Variations

We characterized 512 memory blocks in a DRAM module using the process described in Section 3.6.2. The results are presented in Fig. 3.11, 3.12, and 3.13. Table 3.1 provides a summary of the parameter values used for each characterization.

Variation of bit-flips with refresh-pause interval As described in the previous sections, reconfigurability is achieved in the presented design by modifying the refresh-pause interval. Fig. 3.11(a) and 3.11(b) show the variation of bit-flips with refresh-pause interval across 512 blocks in a DRAM module. Fig. 3.11(a) depicts the number

Table 3.1.: Parameter values used in our evaluation

Figure	**Refresh-Pause int.**	**Size**	**Address**	**Wrapper Pattern**
Fig. 3.11	Varied	128 KB	0x38000000 – $\cdots$	All '0's
Fig. 3.12	60 s	Varied	0x38000000 – $\cdots$	All '0's
Fig. 3.13	60 s	128 KB	0x38000000 – $\cdots$	Varied

of bit-flips across different blocks while Fig. 3.11(b) shows the number of blocks having a particular number of bit-flips. Assuming a minimum required entropy of 400 bit-flips, it can be clearly seen that a refresh-pause interval of 20 s is unsuitable for PUF operation. On the other hand, an interval of 60 s generates entropy in excess of 1200 bit-flips across all the blocks while 40 s does it across some of the blocks, hence, both are suitable refresh-pause intervals. The entropy variation across suitable intervals is quite high, thus, reinforcing our intuition for the use of refresh-pause interval as the parameter for reconfiguration. Moreover, Fig. 3.11 provides us the minimum interval (for the given DRAM module) that meets the entropy requirements. As compared to Ref. [11], which uses a refresh-pause interval of 256 s for a minimum required entropy of 512 bit-flips, our choice of the minimum interval (60 s) reduces the authentication time by `4.3X` (256/60=4.3) while meeting the same entropy requirement. Note that the minimum interval may vary from one module to another, *e.g.,* meeting the same entropy requirement in some of the other modules required a minimum interval of 40 s, thus, further reducing the authentication time by `6.4X` (256/40=6.4).

Variation of bit-flips with block-size Blocks of different sizes starting at the same address can contain widely varying bit-flips. Fig. 3.12(a) and (b) show the variation of bit-flips with block-size across 512 blocks in a DRAM module. The block-size, which is a part of the challenge message, can be varied to generate more CRPs for authentication. Also, for a relatively resilient (to bit-flips) DRAM module, a designer may need to use higher block-sizes at a given refresh-pause interval to meet

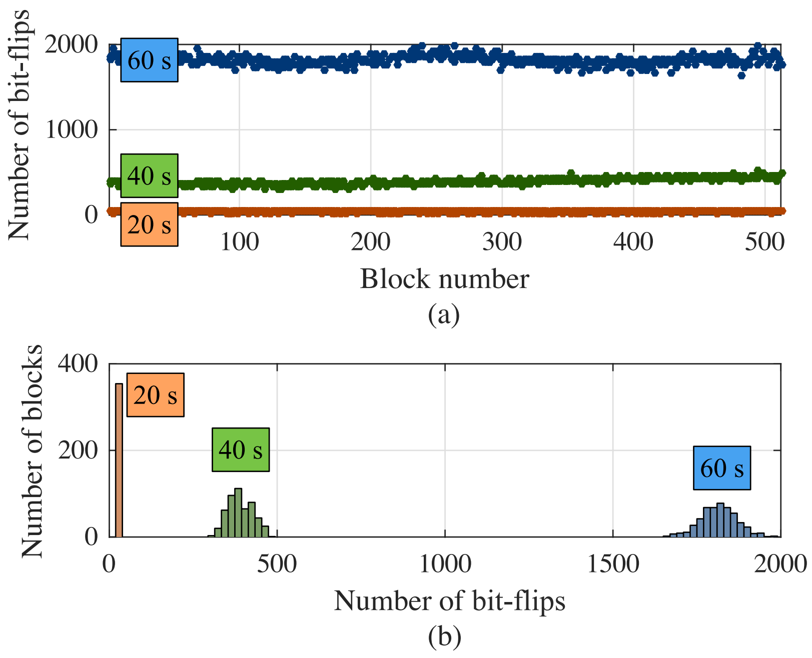


Fig. 3.11.: Variation of bit-flips with refresh-pause interval

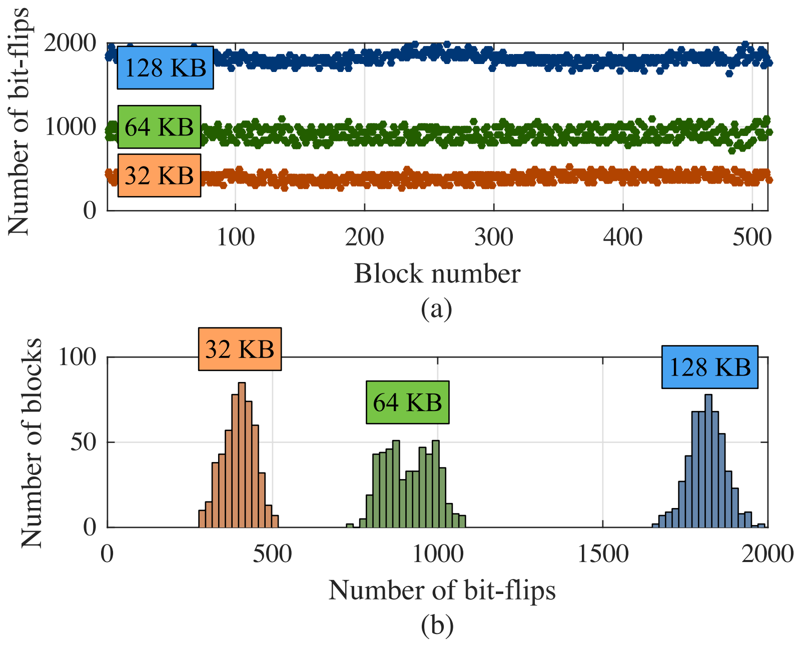


Fig. 3.12.: Variation of bit-flips with block-size

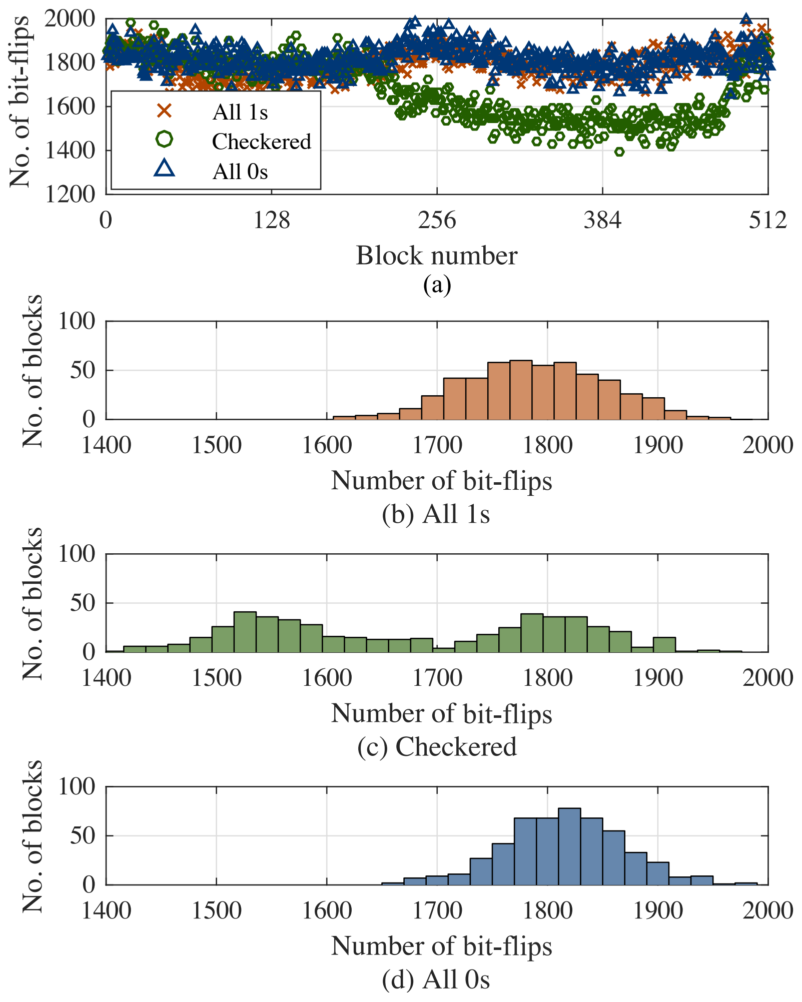


Fig. 3.13.: Variation of bit-flips with wrapper pattern

the minimum entropy requirements. For example, an entropy of 1000 bit-flips is met by all 128 KB blocks and some 64 KB blocks but by none of the 32 KB blocks, as shown in Fig. 3.12. Characterization provides us with valuable insights for choosing the minimum block-size for a given interval.

Variation of bit-flips with wrapper pattern An interesting observation in the characterization results is the variation of bit-flips in a block with the wrapper data (or pattern) surrounding the block. Fig. 3.13 shows the variation of bit-flips with wrapper pattern across 512 blocks in a DRAM module. Wrapper pattern serves as an

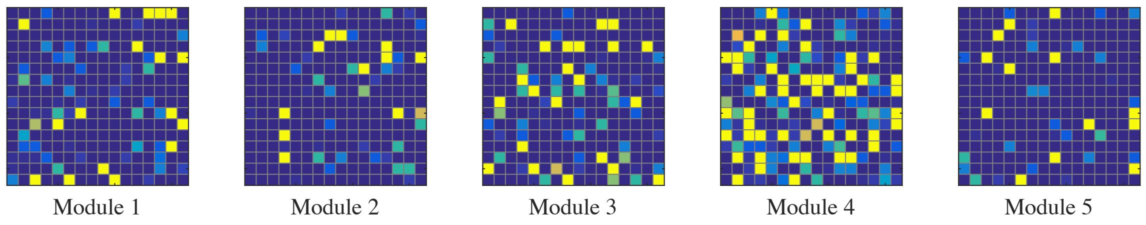


Fig. 3.14.: Unique responses generated from different DRAM modules[1]

additional parameter that can be varied to extract more entropy (and more CRPs) from a DRAM module. However, not all blocks respond to the variation in wrapper pattern, as is evident in the first half of Fig. 3.13(a). Thus, a careful block-specific utilization of this parameter is required. An off-shoot of this analysis is the revelation of the pattern that generates maximum bit-flips (all '0's here), enhancing the entropy of the block.

3.7.3 Uniqueness Analysis

To demonstrate the uniqueness offered by D-PUF, we generated response bitstreams (or fingerprints) from five different DRAM modules using the same refresh-pause interval and other parameters. This is shown in Fig. 3.14, where different colors represent the degree of variation in the number and position of bit-flips in the generated bitstreams. A dark (blue) color implies a small number of bit-flips in the corresponding location within a block. Fig. 3.15 gives a pictorial representation of the different responses generated using the same refresh-pause interval and other parameters (except address) from different blocks belonging to a single DRAM module. It can be seen that the responses are not only unique across different modules but also across different blocks within a module. This goes to show the viability of D-PUF for authentication.

[1]Address = 0x8000000, Size = 128 KB, Wrapper pattern = All '0's, Refresh-pause int. = 90 s; Average HD = 216, Minimum HD = 113

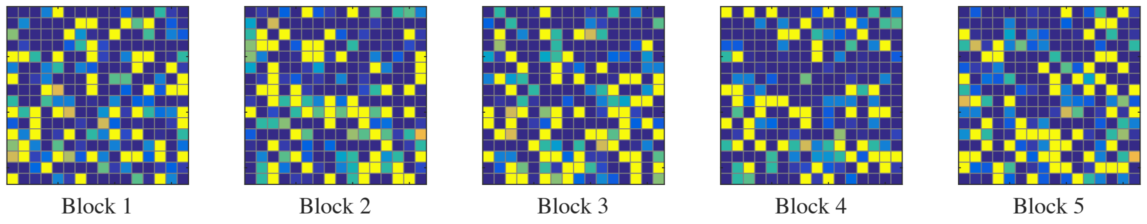


Fig. 3.15.: Unique responses generated from the same DRAM Module[2]

3.7.4 Robustness Analysis: Authentication under Varying Operating Conditions

The reproducibility of responses (to the same challenge) under varying operating conditions such as temperature and aging is important to PUF operation and is referred to as robustness. We quantify robustness as the average HD resulting from the comparisons between golden responses and the responses generated by D-PUF during authentication. These comparisons are referred to as *same-module* comparisons. Also, to demonstrate the uniqueness of responses generated from different DRAM modules, we compared the golden response generated by one module (for a particular challenge) with the corresponding responses (for the same challenge) generated by all the other modules during authentication. We refer to such comparisons as *different-module* comparisons. While *same-module* comparisons produce the true-positive (genuine authentication) rate, *different-module* comparisons give an estimate of the probable false-positive (false authentication) rate.

We performed a robustness analysis of D-PUF under varying environmental and temporal conditions and present our results below. The relative frequency refers to the fraction of total comparisons, either *same-module* or *different-module*, that yields a particular HD.

[2]Module 4; Size = 128 KB, Wrapper pattern = All ‘0’s, Refresh-pause interval = 60 s; Average HD = 503, Minimum HD = 486

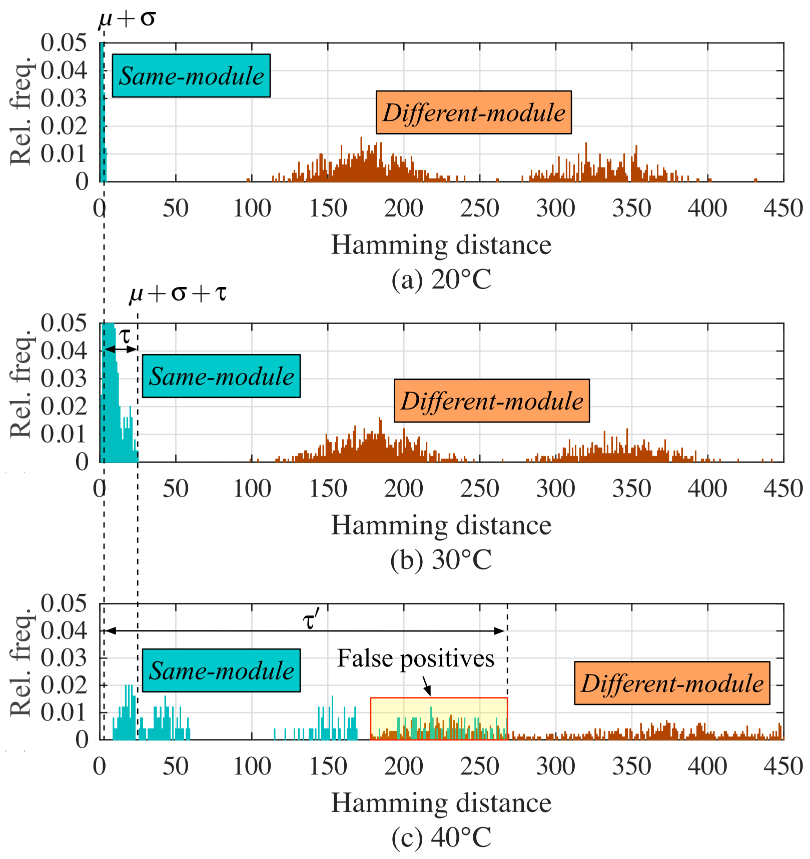


Fig. 3.16.: Robustness across different temperatures (naive approach)

A. Robustness under temperature variations Fig. 3.16 shows the relative frequency versus HD corresponding to 50 different challenges, each applied to five different DRAM modules at three different temperatures. The depicted results correspond to the naive approach comprising static MT and inadequate error correction, both of which are explained in the following paragraphs. The golden responses were generated at 20°C (during *enrollment* phase) while the responses for authentication were generated at 20°C, 30°C, and 40°C (during *authentication* phase). A total of 250 *same-module* comparisons and 1000 *different-module* comparisons were made altogether for the five modules. As shown, the maximum HD for *same-module* comparisons was less than 27 at 30°C but rose exponentially at 40°C. Setting an MT of 27 authenticated all the five modules at 30°C for every challenge. However, at 40°C this led to a number of false-negatives. Hence, the design utilizing the naive approach is robust under a temperature variation of ±10°C. Note that this temperature range also depends on the error-correction algorithm utilized, which happens to be (31,26) Hamming Encoder/Decoder in our case. Beyond this range, either re-enrollment needs to be performed at the new temperature and/or a more powerful error-correction algorithm needs to be used. The HD margin between the *same-module* comparisons and the *different-module* comparisons re-emphasizes the uniqueness of the responses generated by the different modules. The margin also plays an important role in setting the appropriate value for MT at a particular temperature, as is described below.

Setting match threshold (MT) Section 3.4.5 defines μ, σ, and τ that are used for determining MT. μ and σ correspond to the temperature at which enrollment happens (20°C in our case), as shown in Fig. 3.16(a). At 20°C, $\mu + \sigma$ was observed to be ~ 2. τ represents the maximum allowable HD between responses (golden responses and responses during authentication) due to temperature variations only and guarantees that authentication is carried out with a high true-positive rate but, more importantly, 0% false-positive rate. In the naive approach, we consider τ to be a fraction of the average (over the five representative DRAM modules) minimum bit-flips observed in a block at the enrollment temperature and set it at 25% of the same. Hence, the MT equals 27 ($= 2 + 25$) for an average minimum entropy of 100 bit-flips observed at 20°C. This value of τ, in turn, allows us to determine the temperature range within which the CRP database generated at a particular temperature is valid. For the choice of (31,26) Hamming Encoder/Decoder, this temperature range came out to be close to ±10°C when averaged over five different modules, as depicted in Fig. 3.16. This enabled us to use the same CRP database (generated at 20°C) for 30°C too, achieving a 100% true-positive rate while also ensuring 0% false-positive rate. However, aggressively setting the τ value for incorporating higher temperature ranges may lead to false positives during authentication, as shown in Fig. 3.16(c), and should be avoided.

From the previous discussion, it can be concluded that enrollment at 20°C will enable successful authentication at any temperature between 10°C and 30°C. However, D-PUF may be expected to operate across a much wider temperature range, *e.g.,* 10°C to 60°C. Hence, re-enrollment was performed at 50°C to cater to the remaining temperature range (40°C to 60°C). Authentication was carried out at 50°C and 60°C; the results (from *same-module* comparisons) for module 4 are shown in Fig. 3.17. As in the previous authentication experiment, we also subjected the other four DRAM modules to the same challenges and compared their responses with the corresponding golden responses of module 4 at each of the two temperatures (*different-module* comparisons). Following the same approach towards setting MT, $\mu + \sigma$ came out to be 40 for 50°C while τ was calculated to be equal to 1300 for 60°C (average number of bit-flips observed at 50°C across the representative modules was 5200). Hence, MT equals 40 and 1340 for 50°C and 60°C, respectively. Using these values of MT generated a true-positive rate of 97% at 50°C. However, at 60°C, the true-positive rate was 0%. In fact, even if the MT value were substantially higher, it would have been impossible to achieve more than 3% true-positive rate at 60°C due to the merging of same-module and different-module comparison HDs, as shown in Fig. 3.17. This motivates the need for adequate error correction, which is described next.

Adequate error correction Temperature variations cause the number of bit-flips in a DRAM module to change exponentially. When enrollment and authentication are carried out at different temperatures, this phenomenon poses a particularly serious problem; the response generated at the authentication temperature is substantially different from the golden response generated at the enrollment temperature. If the difference is greater than MT, the authentication outcome is false, even for an authentic module. Error correction comes to the rescue in such a case and suppresses some of the difference. Ideally, for an authentic module, error correction should bring down the difference to below MT so that it is successfully authenticated. But, as shown in Fig. 3.17, the error correction ((31,26) Hamming Encoder/Decoder) in module 4

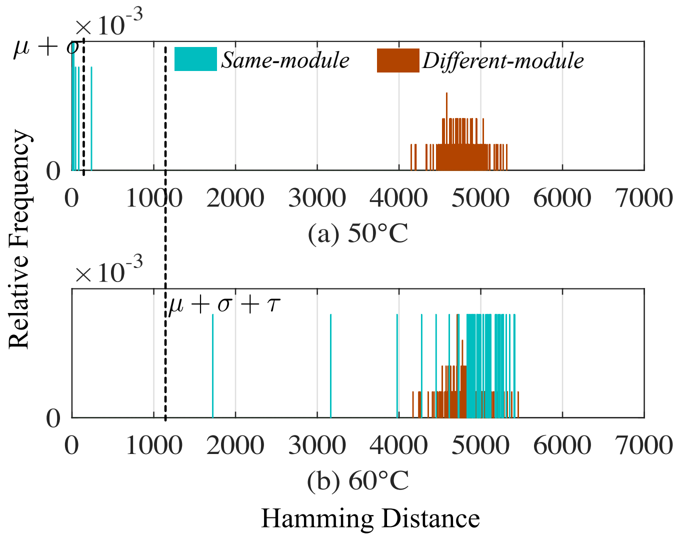


Fig. 3.17.: Inadequate error correction

is not adequate to bring the *same-module* comparisons HD to below 1340 (MT) at 60°C, thus, generating a true-positive rate as low as 0%. However, the same error correction worked very well at lower authentication temperatures (20°C and 30°C). This is because the number of bit-flips at lower authentication temperatures is substantially smaller than that at higher temperatures (50°C and 60°C). As a result, the difference between the golden response and response during authentication, which the error correction needs to suppress, is also small. Note that the error correction should be adequate to handle differences across the complete operating range. In this work, we call the error correction adequate if it can bring down the difference to less than a certain percentage (*error percentage*, set by the designer) of the total bit-flips observed at the highest authentication temperature. Alternatively, instead of having adequate error correction, one may just increase the MT value to improve the true-positive rate at higher authentication temperatures; but, this will also generate false-positives and is, hence, unacceptable. So, a relatively lower value of MT should be used which, in turn, is enabled by adequate error correction. To show the benefits of adequate error correction, we repeated the authentication experiment with

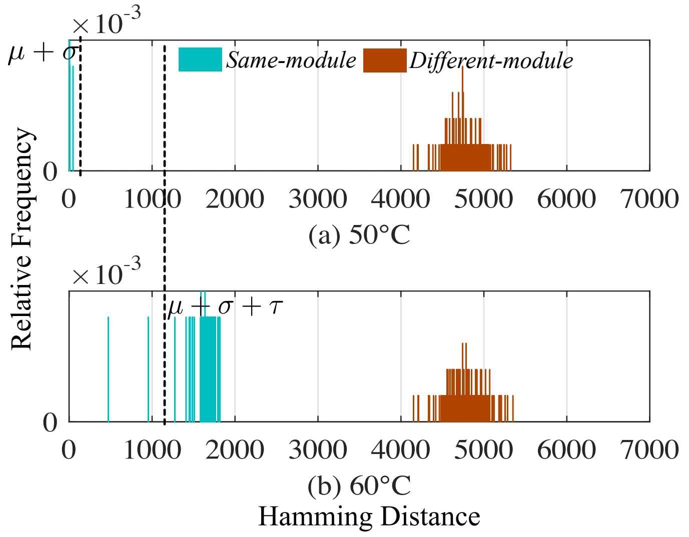


Fig. 3.18.: Adequate error correction with static MT

module 4 by setting *error percentage* equal to 5% and selecting (15,11) Hamming Encoder/Decoder for error correction, accordingly. Fig. 3.18 shows the distribution of *same-module* and *different-module* comparisons HD. With the same MT values, we achieve a true-positive rate of 100% at 50°C but still, a relatively low 30% at 60°C. Further improving the true-positive rate requires setting the *match threshold* dynamically with temperature, as explained in the following paragraph.

Dynamically setting match threshold Till now, MT was set statically, based on the average number of bit-flips observed in a few representative modules at the enrollment temperature. In other words, the MT value was invariable to the actual authentication temperature. However, this can lead to very low true-positive rates at some temperatures even with adequate error correction, as was shown in Fig. 3.18. This motivated us to employ a dynamic approach towards setting MT, where τ (in MT) is a function of the number of bit-flips at the current authentication temperature rather than the enrollment temperature. Moreover, for a given module, instead of the average bit-flips observed across different representative modules, the τ value is now based on the bit-flips observed in only that module. To understand this clearly, assume that adequate error correction is already in place and is capable of bringing down the differences (between golden responses and responses during authentication) to less than $P\%$ (*error percentage*) of the total bit-flips observed at the highest authentication temperature. Also, for a given enrollment temperature ($\mathrm{T_E}$), assume that the authentication temperature ($\mathrm{T_A}$) can only vary from $\mathrm{T_E}$-10°C to $\mathrm{T_E}$+10°C. In the dynamic approach, for a given block in a module, we set the τ value (τ_1) to $(P + 2)\%$ of the total bit-flips observed in the block at $\mathrm{T_E}$-10°C for the temperature range [$\mathrm{T_E}$-10°C, $\mathrm{T_E}$). Similarly, we set the τ value (τ_2) to $(P + 4)\%$ of the total bit-flips observed in the block at $\mathrm{T_E}$+10°C for the temperature range ($\mathrm{T_E}$, $\mathrm{T_E}$+10°C]. Note that the percentage values (2% and 4%) were set empirically from experimental observations. Moreover, when $\mathrm{T_A} = \mathrm{T_E}$, we also define τ_0 and set it to $P\%$ of the total bit-flips observed in the block at $\mathrm{T_E}$. This is done to counter the exponentially increasing number of random bit-flips (due to VRT, Section 3.5) with temperature. As before, the final MT values at different temperatures are generated by adding the τ values with the corresponding μ and σ values (the latter two are a function of the module and the enrollment temperature).

We show the benefits of this new approach in Fig. 3.19. Note that each of the modules was subjected to the same challenges and at the same temperatures as the previous authentication experiment. In addition, authentication at 40°C was also performed. We used a (15,11) Hamming Encoder/Decoder for error correction that is capable of bringing the difference to less than 6% (*error percentage*) of the total bit-flips at the highest authentication temperature (60°C) across every module. Accordingly, τ_0, τ_1, and τ_2 were set to be 6%, 8%, and 10%, respectively, of the total bit-flips observed at the corresponding temperatures (50°C, 40°C, and 60°C, respectively). Note that error correction can also be a function of the module; a module that exhibits small number of bit-flips need not utilize very powerful error correction. To simplify our experimentation methodology, we utilized the same error correction across all the modules. As an example, the MT values at different temperatures have been marked for module 4 in Fig. 3.19. The dynamic approach toward setting MT,

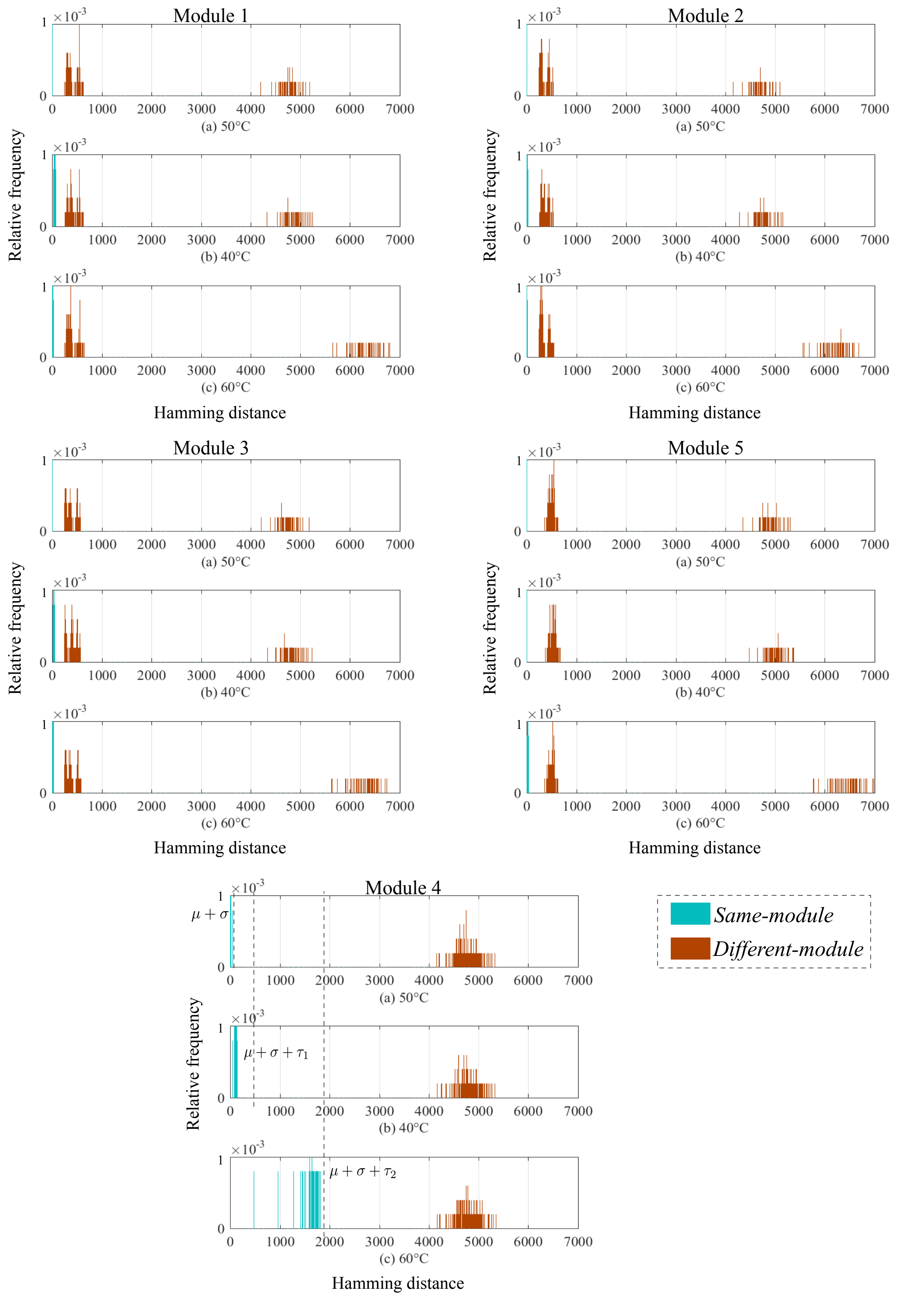


Fig. 3.19.: Robustness across different temperatures (dynamic MT with adequate error correction)

coupled with adequate error correction, enabled us to achieve a 100% true-positive rate without any false-positives at higher authentication temperatures for every module. Experiments also showed that this approach is equally effective for authentication at relatively lower temperatures as we achieved 100% true-positive rate without any false-positives at 10°C, 20°C, and 30°C. Thus, the operating range of D-PUF for device authentication is substantially improved using the dynamic approach.

B. Robustness under aging DRAM cells undergo wear and tear over time, and this could adversely affect the authentication process. Thermal stress accelerates the wear rate and could be intentionally applied to a DRAM module to approximate the wear and tear that would happen naturally over a long period (of time). We use this technique to approximate ten months of wear and tear (or aging) in one of the DRAM modules by subjecting it to a temperature of 85°C for 48 hours. The Arrhenius equation [68], typically used for predicting reliability/failure rates, is then used to calculate the acceleration factor (AF) resulting from the applied thermal stress. As per the Arrhenius equation,

$$AF = e^{\left(\frac{E_a}{k}\left(\frac{1}{T_{\text{use}}} - \frac{1}{T_{\text{stress}}}\right)\right)} \tag{3.1}$$

where,

AF = Acceleration Factor

E_a = Thermal Activation Energy (value used = 0.7 eV [69])

k = Boltzmann's Constant (8.63 x 10^{-5} eV/K)

T_{use} = Use Temperature (value used = 293 K (20°C))

T_{stress} = Stress Temperature (value used = 358 K (85°C))

Using these values, AF was calculated to be 152. Since the thermal stress was applied for 48 hours, this translates to 7296 (152×48) hours or ten months of natural aging (at 20°C).

Fig. 3.20 gives the relative frequency (of *same-module* comparisons) versus HD before and after aging of a DRAM module for 50 different challenges and at a constant

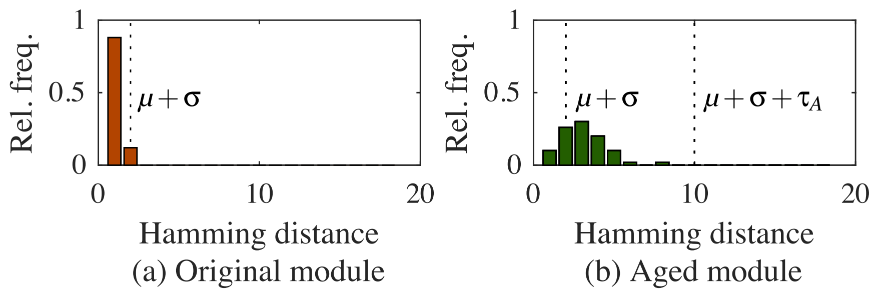


Fig. 3.20.: Robustness under aging

temperature of 20°C. Aging does not affect the bit-flip behavior of DRAM modules much, and this allowed us to use a static approach similar to what was used earlier to address temperature variations. We define τ with respect to aging (τ_A) and set the MT equal to 10. Note that the enrollment was done at 20°C before the aging process was carried out. As shown, aging seemed to generate much smaller HD as compared to temperature variations. Setting an MT of 10 was enough to successfully authenticate the module for every challenge.

3.7.5 True Random Number Generation

As described in Section 3.5, we exploit the *random bit-flips* in DRAM cells, caused due to VRT, for generation of true random numbers. Fig. 3.21 shows the variation of *random bit-flips* with refresh-pause interval and temperature across 256 blocks in a DRAM module. Each of the blocks is 128 KB in *size* and was subjected to a checkered (alternating 0 and 1) input *bitstream* and a checkered *wrapper pattern*. To generate each graph, sixteen runs (spread over 24 hours) were carried out across each of the blocks using the refresh pausing approach. The number of *random bit-flips* in each block was then calculated by comparing the read out bitstream from different runs. Note that the bitstream was directly read out from the DRAM after the specified refresh-pause interval, *i.e.*, it does not undergo any hashing. The value of *flipProb* was set to 0.5, *i.e.*, a bit was said to exhibit *random bit-flip* if it flipped at least

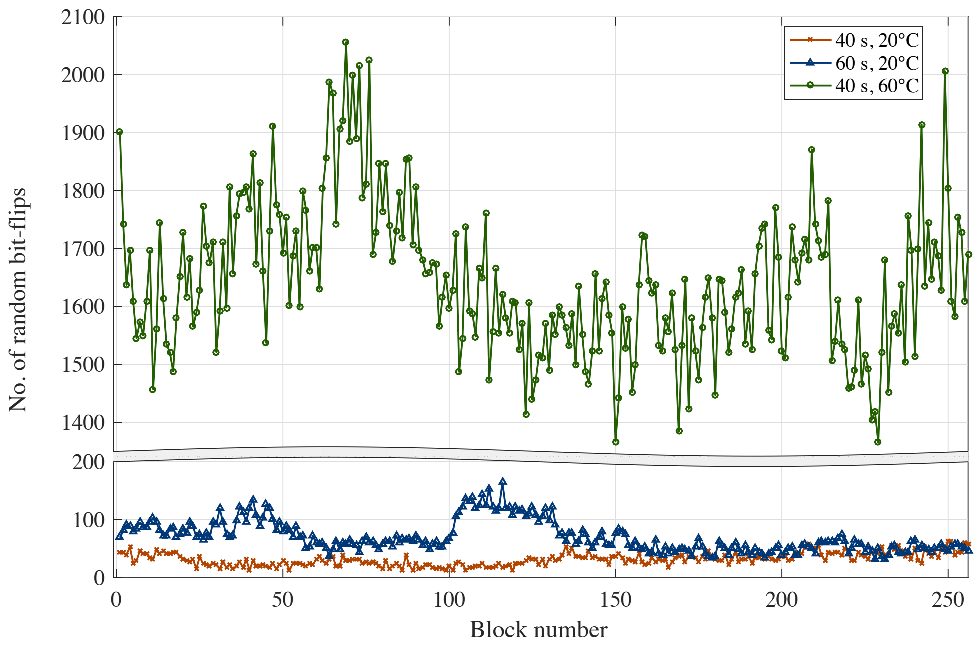


Fig. 3.21.: Variation of *random bit-flips* with refresh-pause interval and temperature

eight times (across the sixteen runs) with respect to the input bit-value. As shown in Fig. 3.21, the number of *random bit-flips* varies across different blocks, hence, selecting suitable blocks using the characterization results is an important step for random number generation using D-PUF. Also, at 20°C, increasing the refresh-pause interval from 40 s to 60 s increases the number of random bit-flips for blocks 1-160, however, blocks 161-256 remain unaffected. This interesting observation stresses the importance of the characterization process. Specifically, one should avoid performing reconfiguration by changing the refresh-pause interval (reconfiguration parameter) from 40 s to 60 s for blocks 161-256; it does not substantially alter the random bit-flip behavior in these blocks. Fig. 3.21 also shows the effect of temperature on the number of random bit-flips. As expected, moving from 20°C to 60°C (with all parameters remaining constant) increased the number of random bit-flips by ~20 times. The high dependence of bit-flip (both random and deterministic) behavior on temperature, though an impediment for device authentication, works favorably for random

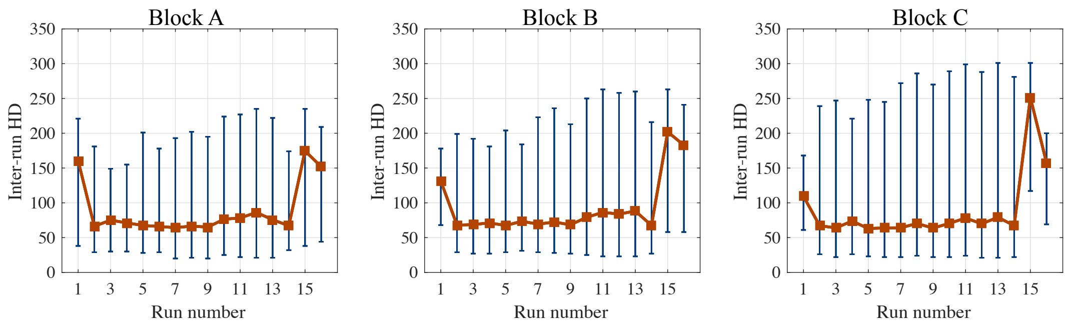
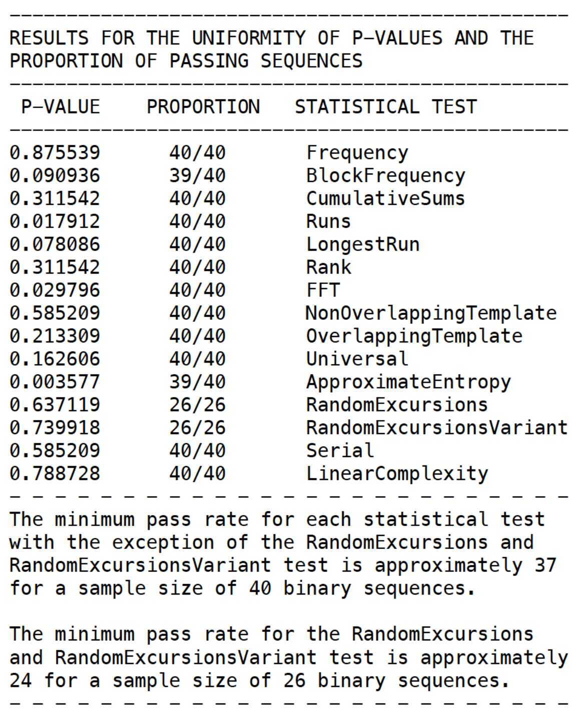


Fig. 3.22.: Inter-run HD across 16 runs spread over 24 hours

number generation. Hence, when operating at high temperatures, a relatively lower refresh-pause interval can be used to generate a random number, thus, substantially increasing the throughput of the random number generator.

To highlight this randomness, in Fig. 3.22, we show the inter-run HD of the read-out bitstreams (generated in the previous experiment) for three different blocks with 40 s refresh-pause interval and at 20°C temperature. This gives an approximate measure of the difference in bit-flips that can be expected across two iterations of the random number generation process. For every block, the bit-stream in each run is compared against that of every other run. This generates an average value and range, in terms of HD, which are represented by a line and various vertical bars, respectively, in Fig. 3.22. The minimum inter-run HD is ~25 in each block, i.e., there are at least 25 bits that flip randomly across any two runs. Note that this minimum difference depends on the parameter values used, temperature as well as the block (and the DRAM module) itself.

The TRNG (in D-PUF) was verified using the NIST Statistical Test Suite [70]. National Institute of Standards and Technology (NIST) specifies a set of 15 statistical tests that are useful in evaluating the randomness of arbitrarily long binary sequences, produced by software or hardware generators. As described earlier, we first generate a random bitstream from a block using the refresh pausing approach and subsequently hash the bitstream to produce the random number. We utilize SHA-256 for hashing



```
------------------------------------------------
RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE
PROPORTION OF PASSING SEQUENCES
------------------------------------------------
 P-VALUE    PROPORTION   STATISTICAL TEST
------------------------------------------------
0.875539     40/40       Frequency
0.090936     39/40       BlockFrequency
0.311542     40/40       CumulativeSums
0.017912     40/40       Runs
0.078086     40/40       LongestRun
0.311542     40/40       Rank
0.029796     40/40       FFT
0.585209     40/40       NonOverlappingTemplate
0.213309     40/40       OverlappingTemplate
0.162606     40/40       Universal
0.003577     39/40       ApproximateEntropy
0.637119     26/26       RandomExcursions
0.739918     26/26       RandomExcursionsVariant
0.585209     40/40       Serial
0.788728     40/40       LinearComplexity
- - - - - - - - - - - - - - - - - - - - - - - - -
The minimum pass rate for each statistical test
with the exception of the RandomExcursions and
RandomExcursionsVariant test is approximately 37
for a sample size of 40 binary sequences.

The minimum pass rate for the RandomExcursions
and RandomExcursionsVariant test is approximately
24 for a sample size of 26 binary sequences.
- - - - - - - - - - - - - - - - - - - - - - - - -
```

Fig. 3.23.: NIST Statistical Test Suite results

in our design, thus generating a 256-bit random number in every iteration. Alternatively, other hashing algorithms could also be used. For generating the bit-streams, we used a 4 MB block (selected through characterization) in a sample DRAM module and subjected it to a checkered input *bitstream*, checkered *wrapper pattern*, and 40 s refresh-pause interval. The temperature was set at 20°C. In the NIST Test Suite, the minimum length of each sequence to be tested varies with the individual tests; longer sequences generate more statistically accurate results. Hence, we generated 40 sequences that are each 1,007,616-bits long and satisfy the length requirements for every test. A sequence consisted of 123 iterations of the refresh pausing operation. Each iteration, in turn, consisted of 8192 bits and was generated using the method shown earlier in Fig. 3.8. The generated bits from each iteration were finally concatenated to form one full sequence. Note that while all the iterations in a particular sequence were executed immediately one after the another, the 40 sequences were not generated consecutively; the time-gap between generating one sequence and the next was varied from hours to days. Fig. 3.23 summarizes the test outcomes of the 40 sequences. The first column, P-VALUE, shows the uniformity of the *P-values* [70] calculated in each test across the 40 sequences. A P-VALUE greater than or equal to 0.0001 represents uniform distribution, and the test is considered to be passed. The second column, PROPORTION, specifies the ratio of the number of sequences that pass the individual test to the total number of sequences. If this ratio is greater than or equal to the threshold value specified at the bottom of Fig. 3.23, then the test is passed. Note that some of these tests consist of multiple sub-tests, each of which is passed by the TRNG in D-PUF. However, due to space constraints, we only show a representative subset of the results. The third column, STATISTICAL TEST, specifies the individual test being applied. In summary, the numbers generated by D-PUF using the refresh pausing approach are truly random in nature and pass all the tests specified by NIST.

3.8 Discussions

This section briefly discusses some additional design aspects of D-PUF.

3.8.1 Attack Scenarios and Assumptions

Before describing our attacker model, we state our assumptions for D-PUF (P). First, the responses generated by P are assumed to be unpredictable, and P itself is unclonable. The randomness in bit-flips coupled with the reconfigurability of P validate this assumption. Second, the characterization results of P are only shared with the authenticator in a secure environment and are inaccessible to an external entity.

We envision three major types of attack that could be mounted on P – snooping-based, physical invasion-based, and replay attacks. The most probable type, *i.e.,* snooping-based attack, happens when an attacker learns a subset of the CRPs corresponding to a particular block and refresh-pause interval, by passively listening to the communication between D and A during the *authentication* phase. The attacker could then employ sophisticated techniques such as machine learning [14] to predict the behavior of P. However, the probability of success for such an attack is very low due to the following reasons. First, if a different block is used for subsequent authentications, it is near impossible for an attacker to predict the response of the block based on the response of a known one due to random yet unique bit-flips across different blocks (Fig. 3.15). Second, if the PUF is reconfigured, it becomes even more difficult to predict the behavior of P. To explain this, we refer to Fig. 3.11. Suppose, passive snooping allows the attacker to have complete information about the number, position, and nature of all the bit-flips in a certain block (128 KB size) in P for the 40 s refresh-pause interval. The same block when used during another *authentication* phase, employing 60 s interval, would generate ∼1400 new bit-flips. So, the probability of predicting the correct response is extremely low (close to $\frac{1}{^{128\times1024\times8}C_{1400}}$), given the limited number of times the attacker can request authentication. Third, even if the same block is continued to be used for authentication, such an attack could be prevented by generating (and using) challenges (Section 3.8.3) that utilize some minimum *unique* entropy (or bit-flips).

The second type of attack requires a highly-skilled and well-equipped attacker. Through physically invasive means, he/she can access the DRAM module in D and characterize it to learn about all the possible bit-flips in the module. However, such an attack is unlikely due to the following reasons. First, the attacker needs to possess the DRAM module for a sufficiently long period of time to be able to characterize it exhaustively. In the worst-case scenario, he may be able to characterize a portion of the DRAM for a few values of the parameters. But, the PUF can be easily reconfigured preventing the attacker from predicting the DRAM behavior completely. Second, though the attacker may extract the helper data (usually stored in the public NVM of D) that leaks some information about the expected response, it is unusable as each challenge (and the corresponding response and helper data) is used only once during authentication.

Finally, replay attacks cannot be mounted on a PUF as a CRP is never used twice during authentication.

3.8.2 Mitigation of Authentication Latency

Sometimes, D-PUF may not be able to generate the response for a challenge instantaneously due to a high refresh-pause interval or non-availability of a sub-address space (due to the sharing of DRAM among multiple applications). One potential way of mitigating this latency is by piggybacking future challenges with the current one and caching the corresponding responses within *D*, which could be then sent during the next authentication cycle.

3.8.3 Generation of Challenge Bit-Stream

The randomly generated challenge may contain a *bitstream* such that none of the bits flip during response generation. For example, a DRAM block containing only *true*-cells (‘1’ $\rightarrow$ ‘0’ bit-flips), when subjected to a challenge *bitstream* containing all 0s, would generate a response *bitstream*, which is same as that of the challenge. To prevent this, the challenge *bitstream* can be constructed with the help of the characterization results obtained prior to enrollment. By putting `‘1’` and `‘0’` at the *true*-cell and *anti*-cell positions respectively in a challenge *bitstream*, the inherent entropy of the module can be properly utilized.

3.8.4 Block Selection: Solution Space and Optimal Solution

In the *characterization* phase, Algorithm 1 is used to select blocks that meet the entropy requirements, as specified by the PUF designer. Note that the selected blocks could be of varying sizes (Z_1, Z_2, *etc.*) and hence, the solution space could be very large. Here, the solution refers to any list of non-overlapping blocks (could be of varying sizes), each of which satisfies the entropy requirement. Though the sizes of blocks in a solution do not affect the authentication outcome, a solution consisting of mostly large blocks will incur much higher operational overhead as compared to one consisting of relatively smaller blocks. A large block requires longer challenge/response messages, thus increasing the CRP database size (stored at the authenticator) as well as the message transmission time between the device and the authenticator. The energy expended by the device to generate each response will also increase due to write/read operations happening over a large block. Hence, Algorithm 1 tries to find an optimal solution, *i.e.,* a list of non-overlapping blocks that are as small as possible and each of which satisfies the entropy requirement. Accordingly, blocks of smallest sizes are selected first before moving on to larger blocks. In addition to having low operational overhead, an optimal solution selects the maximum number of blocks as compared to any suboptimal solution, thus providing more flexibility to the designer for varying the different parameters associated with the response-generation process. This is also important since the fulfillment of the "selection of a minimum number of blocks (N_{min})" criteria is necessary for the *characterization* phase to end successfully.

3.9 Conclusion

In this chapter, we proposed an *intrinsically reconfigurable* DRAM PUF based on refresh pausing and also presented a secure, low-overhead mechanism that uses the PUF for device authentication. We validated our work on a real system using off-the-shelf DRAM modules and evaluated it thoroughly. The overall design performed robustly under various environmental and temporal variations and achieved a 4.3X-6.4X reduction in authentication time, compared to prior work. The DRAM PUF was also used as a true random number generator and was verified using the NIST Statistical Test Suite. We envision that our work will pave the way for the wide adoption of DRAM PUFs into a large number of modern embedded systems.

4. MEMORY-BASED COMBINATION PUFS FOR DEVICE AUTHENTICATION IN EMBEDDED SYSTEMS

Current memory-based PUFs are constructed using a single memory component in the device, *i.e.,* based on a single entropy source. This means that the PUF represents the identity of the component and not that of the system. If the component is removed and transferred to a different system (invasive attack), the identity transfers over as well, which is undesirable. Moreover, the use of a single memory component makes it vulnerable to sophisticated non-invasive attacks [14]. To mitigate these concerns, it is desirable that the PUF be dependent on multiple system components (some of which may be more tightly integrated into the system than others), thereby performing *multi-component authentication.* For example, Fig. 4.1 shows several IoT devices containing multiple memory technologies or components. Besides using a single entropy source, several memory-based PUFs also suffer from other shortcomings such as low entropy [11, 49] and limited number of Challenge-Response Pairs (CRPs) [43,44,49].

4.1 Chapter Contributions

Recent works [15,16] have tried to address a subset of the above-mentioned shortcomings. However, the choice of entropy sources used in these PUF designs renders them unsuitable for multi-component authentication. They also require the addition of custom hardware to the system and, hence, cannot be implemented using Commercial-Off-The-Shelf (COTS) systems. In this work, we overcome these limitations by proposing the design of a memory-based combination PUF, henceforth referred to as *C-PUF*. By tightly integrating heterogeneous memory technologies, *C-PUF* exhibits high entropy alongside an exponential number of CRPs, and takes

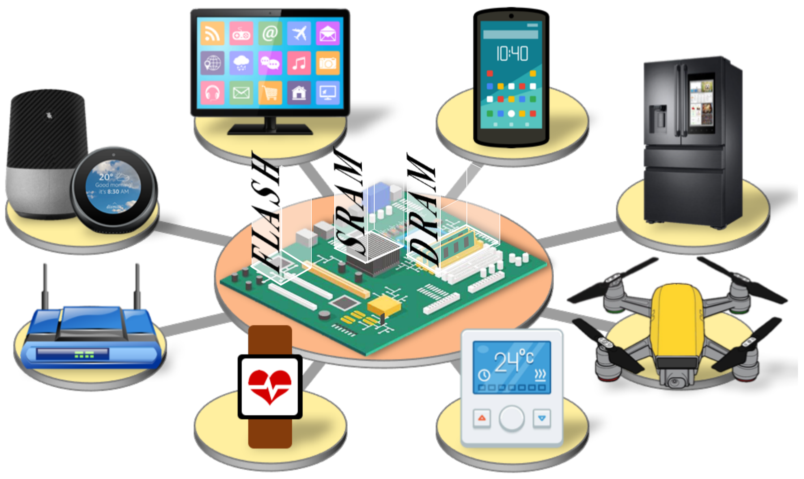


Fig. 4.1.: Examples of IoT devices containing multiple memory technologies

the first step towards multi-component authentication in an embedded device. The heterogeneous nature of the entropy sources (memories) used and *C-PUF*'s ability to undergo *intrinsic reconfiguration* (ability to reconfigure the PUF at runtime without any additional hardware) protects it from various security attacks. *C-PUF* also features a light-weight authentication scheme to ensure robust operation (authentication) under wide environmental and temporal variations. Specifically, we make the following contributions [9] :

- We propose the concept and design of a memory-based combination PUF (*C-PUF*) that tightly integrates heterogeneous memory technologies to construct a PUF. *C-PUF* *(i)* takes a step towards multi-component authentication, *(ii)* exhibits high entropy and supports a large number of CRPs, *(iii)* is intrinsically reconfigurable, and *(iv)* requires minimal or no additional (custom) hardware, hence, can be easily implemented on a COTS device.

- As a key enabler for authentication using *C-PUF*, we propose a lightweight scheme that ensures its robust operation under environmental and temporal variations. We also propose two lightweight algorithms that assist the authentication scheme in performing error correction in the generated responses.

- We implement, demonstrate, and evaluate several fully-functional prototypes of *C-PUF* in a real system using two widely-used memory technologies, Static Random Access Memories (SRAMs) and Dynamic Random Access Memories (DRAMs). Extensive authentication tests performed across a wide temperature range (20°C – 55°C) and accelerated aging (12 months) achieved greater than 97.5% true-positive rate. The absence of any false-positives, even under an invasive attack, further highlights the effectiveness of the overall design.

The rest of this chapter is organized as follows. Section 4.2 describes the motivation behind the work. Next, Section 4.3 presents the details of *C-PUF's* design and also discusses the proposed lightweight authentication scheme. Section 4.4 describes the experimental setup while Section 4.5 shows the results of the experiments that were performed to validate the design. Section 4.6 discusses potential attack scenarios and additional design aspects of *C-PUF*. Finally, Section 4.7 concludes the chapter.

4.2 Motivation

Memory-based PUFs require minimal or no additional hardware for their operation as they use components that are already present in most modern embedded systems. As a result, they can be easily implemented on COTS devices, and hence present a distinct advantage over other PUF implementations. However, current memory-based PUFs suffer from several shortcomings, as described below.

4.2.1 Identity based on a Single Component

Current memory-based PUFs use a single memory component (or technology) in their construction, in other words, they are based on a single entropy source (in the system). This means that a PUF represents the identity of the component and not that of the system. Note that some of these components are often loosely integrated into the system (*e.g.,* using a removable/replaceable DRAM SODIMM). As shown

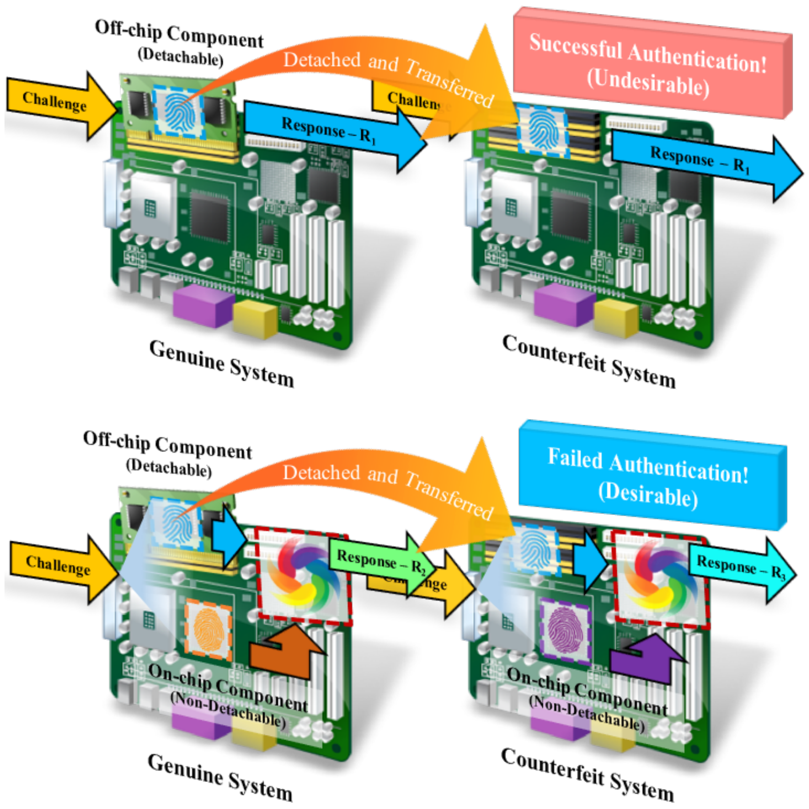


Fig. 4.2.: Authentication using – (a) a Single Component and (b) Multiple Components in a system

in Fig. 4.2(a), an invasive attack may involve (removal and) transfer of one such component to a (different) counterfeit system, transferring the identity as well, and resulting in a successful authentication of the counterfeit system. Moreover, the use of a single memory component in the PUF makes it vulnerable to increasingly sophisticated non-invasive attacks [14, 71].

One way of addressing this shortcoming is through the utilization (or combination) of multiple entropy sources. For example, Ref. [15] mentions a patent that combines (the output of) multiple locations of one or several memory blocks located inside an IC to derive unique keys. By using an address decoder that can permute access order across different memory locations/blocks in a potentially unknown manner, the work claims to increase the resistance of the PUF against invasive attacks. A similar

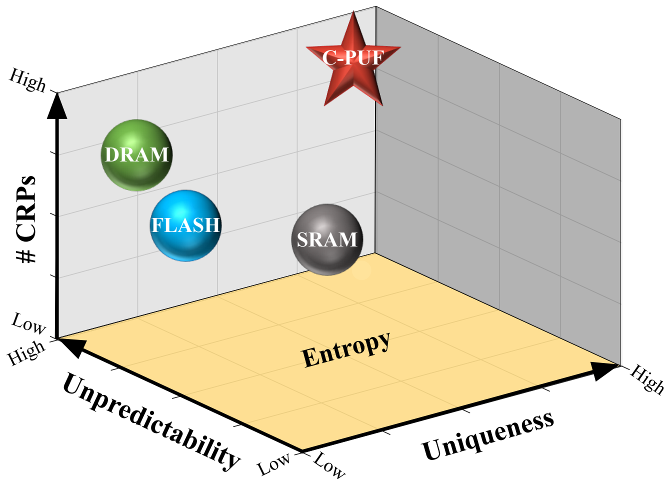


Fig. 4.3.: Qualitative comparison of different memory-based PUFs

but non-memory-based PUF was presented in [16], which proposes combining several on-chip entropy sources (*e.g.,* clock sinks) in an optimized manner towards better (overall) entropy and robustness. However, these work have their own drawbacks, as described in the next section.

4.2.2 Low Entropy and Few Challenge-Response Pairs (CRPs)

To highlight the next shortcoming, we introduce two metrics – *entropy* and *number of CRPs*. In the present context of a PUF, *entropy* [72] manifests itself in the form of – *uniqueness* and *unpredictability*. *Uniqueness* gives a measure of the extent by which one PUF's behavior (response) differs from another (of the same type). *Unpredictability*, on the other hand, represents the probability of incorrectly predicting the behavior of a PUF. Hence, a PUF with high entropy has both high uniqueness and unpredictability. The second metric, *number of CRPs*, represents the total number of Challenge-Response Pairs supported by a PUF. As mentioned earlier, strong PUFs support a large number of CRPs, which makes them well-suited for challenge-response-based authentication. Fig. 4.3 depicts a qualitative comparison of current memory-based PUFs, which were described in the previous section, as per these metrics; a quantitative comparison is presented in Section 4.5.1. As shown, the (*power-cycling*-based) SRAM PUF [43, 44] exhibits high *entropy* but supports a small number of CRPs due to the existence of very few variable parameters (in its challenge-response mechanism) as well the small extent to which these parameters could be varied because of an SRAM's usually small address-space. Thus, the applicability of the SRAM PUF to challenge-response-based authentication is severely limited. On the other hand, the (*refresh-pausing*-based) DRAM PUF, presented in Chapter 3 and prior work [11], employs a challenge-response mechanism involving several widely-variable parameters, supporting a large number of CRPs. However, for practical *refresh-pause intervals* (Section 4.3.1), the entropy (specifically, uniqueness) exhibited by it is much lower than the SRAM PUF. PUFs based on flash memories [49], on the other hand, suffer from both low entropy and support for few CRPs.

These shortcomings in current memory-based PUFs motivate the design of *C-PUF*, which is explained next.

4.3 C-PUF: A Memory-based Combination PUF

C-PUF aims to address the shortcomings of current memory-based PUFs while performing challenge-response-based authentication in a device. Hence, one of the primary motivators behind its design is to (tightly) combine different heterogeneous (memory) components to achieve *multi-component authentication* in an overall system, as depicted in Fig. 4.2(b). Although there are several choices for these components, we utilized two widely-used memory technologies, SRAM and DRAM, in our prototype implementations. The rationale behind this choice stems from the following observation – an SRAM is usually tightly integrated with the processor and located on the same chip/die, whereas a DRAM is usually more loosely integrated (*e.g.,* as an external SODIMM). As a result, using these memories enables *C-PUF* to authenticate both an on-chip (SRAM) and off-chip (DRAM) component, thereby taking a key step towards multi-component authentication in a system. Moreover, the spatial distribution (on-chip and off-chip) of the heterogeneous entropy sources (memories) substantially improves *C-PUF's* ability to resist invasive attacks [73]. Note that an SRAM PUF utilizes the *power-cycling* approach to generate start-up values as responses. On the other hand, a DRAM PUF's responses comprise of unique bit-flip patterns generated through the *refresh-pausing* approach. Instead of simply using them in a standalone manner, in *C-PUF*, we tightly integrate these two approaches, as shown in Fig. 4.4. This integration allows the (challenge-response) behavior of one entropy source to influence that of the other in an unpredictable manner, creating an extremely-complex overall behavior that can provide substantial resistance to *C-PUF* against non-invasive attacks [14, 71]. Note that the *power-cycling* and *refresh-pausing* approaches to SRAM and DRAM PUFs, respectively, were chosen (in the prototype implementations) due to their popularity and wide-adoption in several state-of-the-art systems. The *C-PUF* design and the associated authentication scheme are flexible enough to incorporate other approaches as well [40, 41, 46–48]. In addition to multi-component authentication, the proposed design also enables *C-PUF* to exhibit high entropy and support an exponential number of CRPs (Fig. 4.3), making it particularly suitable for challenge-response-based authentication. Most importantly, all this is achieved while incorporating minimal or no additional hardware, and hence the design can be easily implemented on a COTS device.

As will be clear shortly, the *C-PUF* design utilizes some of the components of D-PUF (Chapter 3), which leveraged the *refresh-pausing* approach in a DRAM for authentication and random-number generation. *C-PUF* builds upon D-PUF but goes further by introducing the concept of and taking a step towards realizing *multi-component authentication*. It also introduces two key-modifications in the DRAM Error Correction stage (compared to D-PUF) – *(i)* tailoring the (strength of the) DRAM error-correction according to the specific operating point (Table. 4.5) and *(ii)* introduction of a new parameter, B_{Max}, to protect against a certain kind of attack that could be mounted on a system containing DRAM in the form a removable/replaceable SODIMM.

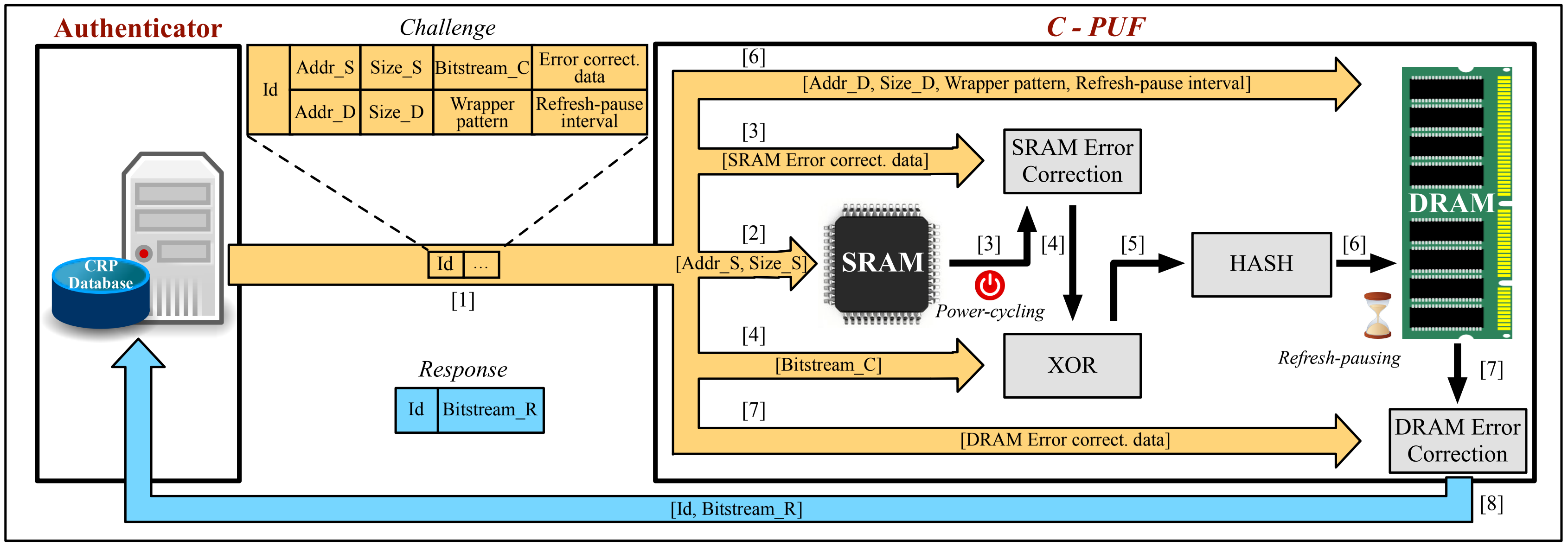


Fig. 4.4.: Overview of the proposed *C-PUF* design utilizing SRAM and DRAM

As mentioned earlier, the idea of utilizing or combining multiple entropy sources has been explored earlier. For example, Ref. [15] combines (the output of) multiple locations of one or several memory blocks located inside an IC to derive unique keys. However, no physical implementation or test results thereof have been published yet. A similar but non-memory-based PUF was presented in [16], which proposes combining several on-chip entropy sources (*e.g.,* clock sinks) in an optimized manner towards better (overall) entropy and robustness. In comparison to these works, the design space for *C-PUF* is much wider as it aims to combine both on-chip and off-chip entropy sources towards realizing *multi-component authentication* in an embedded system. We also propose and validate a low-overhead authentication scheme as part of the *C-PUF* design that ensures robust operation across a wide range of operating conditions. Most importantly, unlike previous work, all this is achieved while incorporating minimal or no additional hardware, enabling easy integration of *C-PUF* into a COTS system.

We now present the design of *C-PUF* in detail, beginning with its challenge-response mechanism.

4.3.1 Challenge-Response Mechanism

C-PUF employs a challenge-response mechanism that utilizes both SRAM *power-cycling* and DRAM *refresh-pausing.* Fig. 4.4 presents the complete sequence of operations associated with this mechanism as well as the formats of the challenge and response. Note that while some parameters in the challenge are SRAM-specific, others are DRAM-specific except *Id*, which represents a unique identifier assigned to a challenge and its corresponding response. These operations are now explained in detail.

[1] First, the authenticator sends a challenge to *C-PUF* in the proper format, as depicted in Fig. 4.4.

[2] The SRAM undergoes *power-cycling* to generate a start-up value of *Size_S* bits from a block beginning at address *Addr_S*.

[3] The start-up value is then corrected for bit-errors in the *SRAM Error Correction* stage with respect to a previously generated *golden* (or expected) start-up value, as described in detail in Section 4.3.2. The information required for this correction is contained in the *Error-correction data* field of the challenge and is generated (prior to this) during the *enrollment phase* (explained later in Section 4.3.4). Note that *Error-correction data* contains information for both SRAM and DRAM error correction. The *SRAM/DRAM Error Correction* stages are responsible for generating their respective error-correction data as well as performing actual error correction; this shall become clearer in the following paragraphs.

[4] The *XOR* stage repeatedly applies the (bit-wise) mathematical operation – *xor* to the corrected start-up value (*CV*), generated in the previous stage, and *Bitstream_C*; *Bitstream_C* is a random binary sequence of *Size_D* bytes and is *xor*-ed across its entire length with *CV*.

[5] The *xor*-ed value then moves to the *HASH* stage, where it is broken down into equally-sized chunks (*e.g.,* 32 bytes), each of which undergoes a mathematical *hash* operation using SHA-256. The output from each chunk is concatenated together to form the complete *hash*-ed value (*HV*). Hashing helps to mask the SRAM start-up value and adds another layer of protection against attacks.

[6] *HV* is applied to the DRAM alongside other parameters *viz. Addr_D*, *Size_D*, *Wrapper pattern*, and *Refresh-pause interval*, to undergo *refresh-pausing*; we follow a similar methodology as described in [8] in this stage. Specifically, the *HV* (of *Size_D* bits) along with the peripheral data, specified by *Wrapper pattern*, is first written onto a block in the DRAM, whose location is specified by *Addr_D*. *Wrapper pattern* is an interesting parameter that was introduced in [8].

It specifies the peripheral data-bits that are written just before the beginning and after the end of the DRAM block, and influences the bit-flip patterns or responses from the DRAM. It can be one of several predefined types, *e.g.,* all '1's, all '0's, checkered, *etc.* Next, refresh operations to the DRAM are paused for a certain amount of time *viz. Refresh-pause interval*, followed by reading of the data (from the same block) containing the bit-flip patterns.

[7] The readout data undergoes error correction in the *DRAM Error Correction* stage to account for the bit-errors caused during the DRAM's response generation. Similar to SRAM error correction, the information required for this correction is also contained in the *Error correction data* field of the challenge and is generated (prior to this) during the *enrollment phase.*

[8] The error-corrected DRAM response, *Bitstream_R*, along with the identifier, *Id*, comprises the (final) *C-PUF* response that is sent back to the authenticator.

As evident, the *SRAM* and *DRAM Error Correction* stages play a pivotal role in *C-PUF's* challenge-response mechanism. We describe these in detail over the next two subsections.

4.3.2 SRAM Error Correction

SRAM start-up values are affected by environmental and temporal variations, which could hinder *C-PUF's* ability to perform authentication successfully. To demonstrate the impact of one such variation *viz.* temperature, we generated start-up values from eight different blocks (32 Bytes in size) each belonging to two different SRAMs and at three different temperatures - 20°C, 40°C, and 55°C. Fig. 4.5 shows the difference, in terms of Hamming Distance (HD), between the start-up values generated at the three temperatures for each of the blocks. The *20°C vs 20°C* comparison involves two different start-up values generated at the same 20°C temperature. It can be seen that the difference is as high as 113 bits ($\sim$44%) between the start-up values gener-

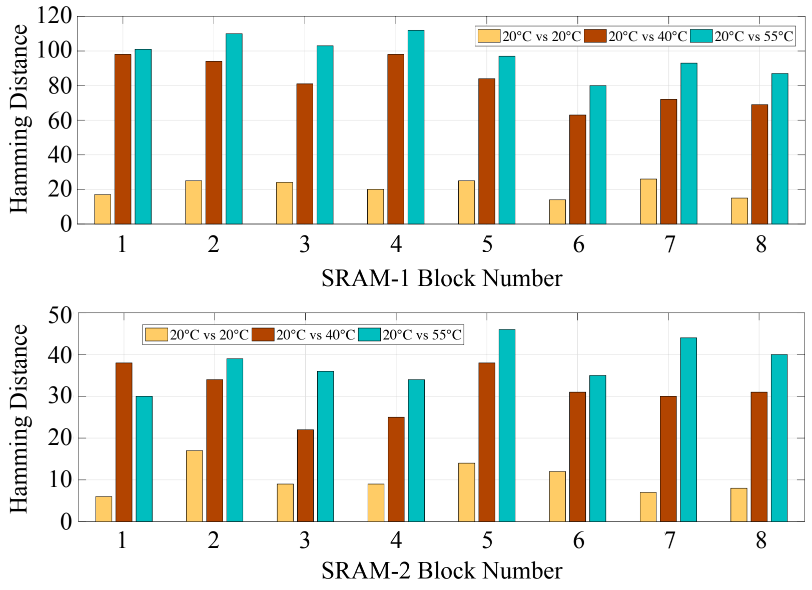


Fig. 4.5.: Variations in SRAM startup values with temperature

ated at 20°C and 55°C for the block numbered 4 in SRAM-1. Hence, temperature variations adversely affect the SRAM's ability to reproduce the same values, which is crucial to performing successful authentication across a wide temperature range. Moreover, the proposed design utilizes a mathematical *hash* (SHA-256) function to scramble the SRAM start-up values and, hence, even a single bit-error could result in a completely different bitstream being subsequently applied to the DRAM (step [6]) in Fig. 4.4). Therefore, the start-up values need to undergo perfect error-correction, *i.e.,* all bit-errors must be corrected before moving on the next-stage (XOR). To keep the SRAM/DRAM error-correction infrastructure simple, we utilize one of the most widely-used error-correction codes in memories *viz.* Hamming Encoder/Decoder (HED). However, we observed that (even n=7, k=4) HED was not sufficiently strong to achieve perfect error-correction in some of the SRAMs across a wide temperature range. Hence, we complement HED with our proposed error-correction scheme, which utilizes minimal computational and storage resources. Specifically, the start-up values from SRAM first undergo error-correction using HED (with n=15, k=11), followed

Algorithm 2: Generation of SRAM Error-correction Data

```
Input: $V_{\text{exp}}$ = Golden (expected) start-up value from SRAM,
$N$ = Number of bits in a segment
Output: $D$ = Error correction data
1  $D = \phi$
2  $S = Get\_All\_NonOverlapSegments(V_{\text{exp}})$
3  foreach $s \in S$ do
4      $repBit_{\text{s}} = 0$
5      if $Num\_Ones(s) \geq Num\_Zeros(s)$ then
6          $repBit_{\text{s}} = 1$
7      $repSeg_{\text{s}} = 0$
8      for $i = 1$ to $N$ do
9          $repSeg_{\text{s}} = (repSeg_{\text{s}} << 1) \mid repBit_{\text{s}};$
10     $D_{\text{s}} = repSeg_{\text{s}} \oplus s$
11     $D = D \cup D_{\text{s}}$
```

by the proposed scheme. The scheme (explained next) comprises of two algorithms, Algorithm 2 and Algorithm 3, that deal with the generation of error-correction data and performing actual error correction, respectively. Empirical analysis showed that the scheme can correct $(N-1)/2$ bits in every N bits (N is an odd integer). We used $N = 7$ in our experiments, and hence achieved a maximum fixable error-percentage of 42.8%. Note that other error-correction schemes could also be utilized; this is orthogonal to the core idea of this work.

4.3.2.1 Generation of SRAM Error-correction Data

Algorithm 2 is utilized by the *SRAM Error Correction* stage to generate the data that is subsequently used for correcting errors in the SRAM start-up values. Note that the error correction data is always generated with respect to the *golden* (expected) start-up value. Algorithm 2 starts by dividing the *golden* start-up value into smaller *segments*; a *segment* comprises of a fixed number of bits, N (= 7, used

Algorithm 3: SRAM Error Correction and Generation of Representative Start-up Value

```
Input: $V_{\text{err}}$ = Erroneous start-up value from SRAM (after Hamming Encoder/Decoder
        correction) to be corrected and converted to $V_{\text{rep}}$,
$N$ = Number of bits in a segment,
$D$ = Error correction data generated from expected (golden) start-up value
Output: $V_{\text{rep}}$ = Representative start-up value
1 $V_{\text{rep}} = \phi$
2 $S = Get\_All\_NonOverlapSegments(V_{\text{err}})$
3 foreach $s \in S$ do
4     $corrSeg_{\text{s}} = D_{\text{s}} \oplus s$
5     $repBit_{\text{s}} = 0$
6     if $Num\_Ones(corSeg_s) \geq Num\_Zeros(corSeg_s)$ then
7         $repBit_{\text{s}} = 1$
8     $V_{\text{rep}} = V_{\text{rep}} \cup repBit_{\text{s}}$
```

here). Each *segment* is then assigned a *representative bit-value* depending upon the relative number of *one*-bits and *zero*-bits in the *segment*. The *representative bit-value* is then expanded to form *representative segment*; the latter is *xor-ed* with the *segment* to generate the *correction data* for that particular *segment*. This data from each *segment* is subsequently combined to form the final error correction data.

4.3.2.2 SRAM Error Correction and Generation of Representative Start-up Value

Algorithm 3 is also utilized by the *SRAM Error Correction* stage and uses the data generated earlier (Algorithm 2) to perform error correction in SRAM start-up values. It starts by dividing the erroneous start-up value (after HED correction) into smaller *segments* of the same size as in Algorithm 2 ($N = 7$ bits). Each *segment* is then *xor-ed* with its respective *correction data* to generate the *corrected segment*. The relative number of *one*-bits and *zero*-bits in a *corrected segment* decides its *representative bit-*

value. All the representative bit-values are then combined to form the *representative start-up value* of the SRAM. Note that this *representative start-up value* is what we refer to as the corrected start-up value (response) of the SRAM throughout the chapter; it is propagated to the *XOR* stage, as shown in Fig. 4.4 (step [4]).

4.3.3 DRAM Error Correction

DRAMs are highly susceptible to environmental and temporal variations. To demonstrate the impact of one such variation *viz.* temperature, we generated DRAM responses from eight different blocks (128 KB in size) each belonging to two different DRAMs and at three different temperatures - 20°C, 40°C, and 55°C. Note that all the DRAM-specific parameters in the challenge were kept constant except for *Addr_D*, which was varied as per the block. Fig. 4.6 shows the difference, in terms of HD, between the responses generated at the three temperatures for each of the blocks. The *20°C vs 20°C* comparison is not shown here as the corresponding HD values are very close to zero. It can be seen that the difference is as high as 5200 bits between the responses generated at 20°C and 55°C for the block numbered 2 in DRAM-1. Thus, the DRAM responses need to be corrected for bit-errors before being sent out to the authenticator as the final response (step [7] in Fig. 4.4). However, unlike the SRAM, perfect error correction is not required as we employ a fuzzy authentication strategy (described in detail in Section 4.3.4) at the authenticator end to determine the outcome of the authentication process. Hence, we utilize just the Hamming Encoder/Decoder (HED) for DRAM error correction in our design. The strength (specified by $<n,k>$) of the HED utilized depends upon the particular DRAM module as well as environmental factors; this is described in detail in the following subsection.

Next, we present the proposed scheme that is utilized for performing authentication using *C-PUF*. An important step associated with this scheme is the setting of appropriate *Match Threshold* value, which determines the (authentication) outcome at the authenticator's end. This is also described in detail in the following subsection.

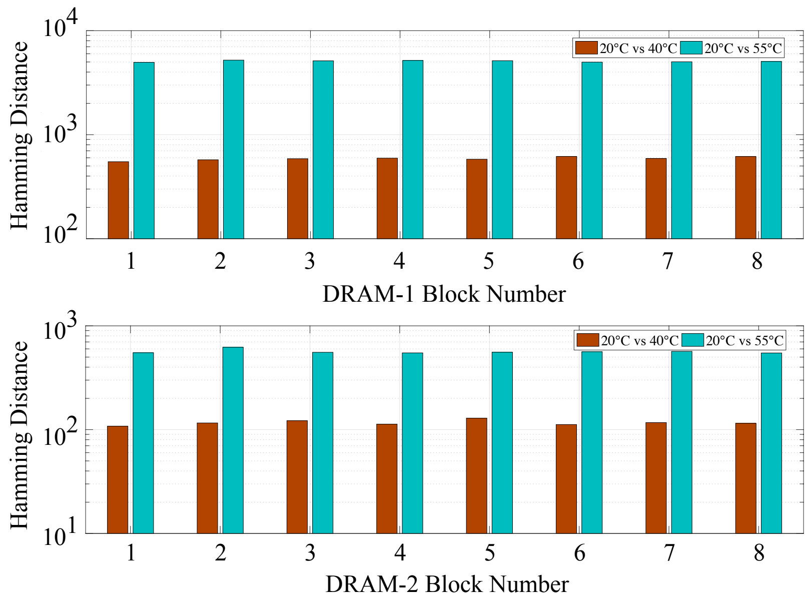


Fig. 4.6.: Variations in DRAM responses with temperature

4.3.4 Device Authentication using C-PUF

Two distinct phases are associated with the proposed authentication scheme involving *C-PUF* – *enrollment phase* and *authentication phase*, as shown in Figs. 4.7 and 4.8, respectively. Both phases utilize the same challenge-response mechanism (described earlier in Section 4.3.1) for response generation but differ in their objectives. The *enrollment phase* primarily deals with the generation of the CRP database by subjecting *C-PUF* to different challenges and recording the generated responses. These responses serve as the *golden responses* (or expected responses); *C-PUF* needs to reproduce them later in order to be successfully authenticated. Also, during this phase, data for subsequent error correction is derived from the *golden* start-up values of the SRAM as well the *golden* responses from the DRAM.

Next, actual authentication of *C-PUF* happens during the *authentication phase*, where it is subjected to the same challenges and is expected to reproduce the *golden responses*. Also, the error-correction data, generated during the *enrollment phase*, is used here to correct the bit-errors in the SRAM start-up values and the DRAM

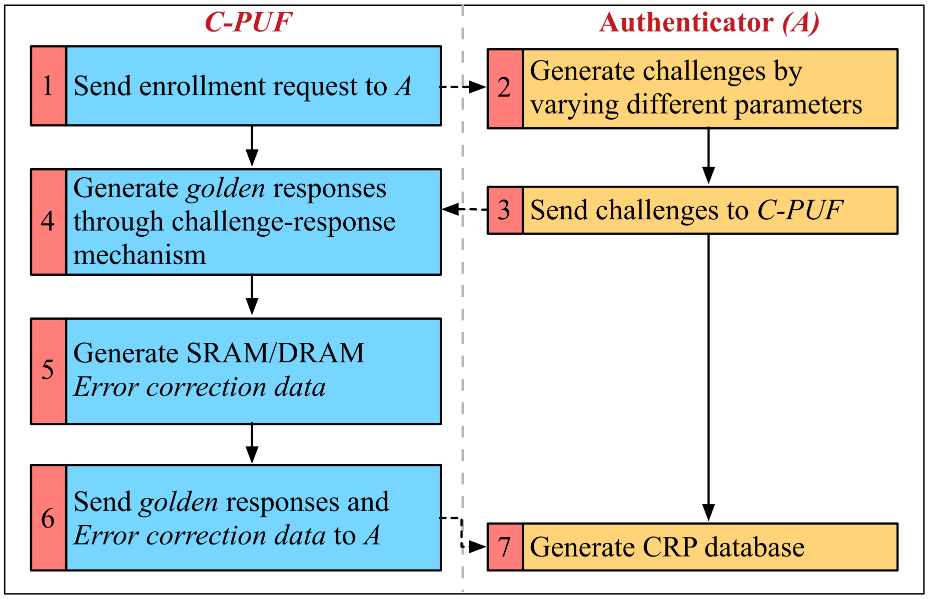

C-PUF
Authenticator *(A)*
1 Send enrollment request to *A*
2 Generate challenges by varying different parameters
3 Send challenges to *C-PUF*
4 Generate *golden* responses through challenge-response mechanism
5 Generate SRAM/DRAM *Error correction data*
6 Send *golden* responses and *Error correction data* to *A*
7 Generate CRP database

Fig. 4.7.: Flowchart showing the *enrollment phase* in *C-PUF*

responses. Finally, the corrected DRAM response is sent to the authenticator, which employs the proposed authentication scheme (based on a fuzzy strategy [8]) to determine the outcome of the authentication process. At the core of this scheme is *Match Threshold* (*MT*), which represents the maximum Hamming Distance (HD) by which a response generated during the *authentication phase* can differ from the *golden response*, and still result in successful authentication. This is particularly relevant in a memory-based PUF since it is affected by environmental and temporal variations and, hence, an exact match of the responses may not happen even if the PUF is genuine. Accordingly, *C-PUF* is successfully authenticated only if this HD is less than or equal to the MT value. Note that *C-PUF* may be expected to operate across a wide range of conditions, *e.g.,* temperature, aging, *etc.* Hence, given an operating range, there are usually multiple points where *C-PUF* undergoes the *enrollment phase*; we refer to these as the *enrollment points* (E_Ps). For example, for temperature, we set the E_Ps at 30°C and 50°C for a sample operating range of [20°C, 55°C]. Similarly, the specific points where *C-PUF* could possibly undergo the *authentication phase* are referred to as the *authentication points* (A_Ps); the set of all A_Ps comprises the (complete)

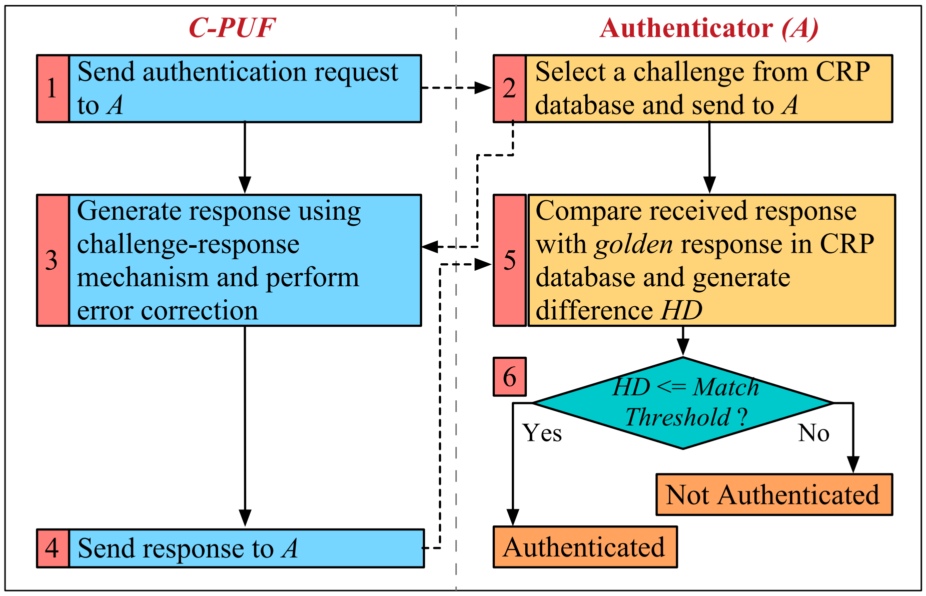

C-PUF
Authenticator (A)
1 Send authentication request to A
2 Select a challenge from CRP database and send to A
3 Generate response using challenge-response mechanism and perform error correction
5 Compare received response with golden response in CRP database and generate difference HD
6 HD <= Match Threshold ?
Yes
No
Not Authenticated
4 Send response to A
Authenticated

Fig. 4.8.: Flowchart showing the *authentication phase* in *C-PUF*

operating range. An important question, however, needs to be answered – *which* E_P *should be chosen for an* A_P *during response comparisons at the authenticator?* We answer this by *mapping* a set of A_Ps to each E_P. In the current implementation of *C-PUF*, all the A_Ps in the range – [20°C, 40°C] and (40°C, 55°C] are mapped to E_P = 30°C and 50°C, respectively. Hence, a response generated during authentication at $A_P = 55°C$ should be compared with the corresponding *golden response* generated at $E_P = 50°C$ to determine the authentication outcome. As mentioned earlier, *Match Threshold* plays a key role in determining this outcome; we describe the mechanism behind setting its value in the following subsection.

Note that, due to the exponentially large CRP space of *C-PUF*, it could be infeasible to store all the CRPs at the authenticator. Hence, we propose that enrollment is performed for a (small) subset of the total CRPs at a time and stored in the CRP database at the authenticator. Once all the challenges in the subset are used-up (for authentication), the *C-PUF* could undergo enrollment again, generating a new subset. Note that the size (number of CRPs) of the subset could be specified by the user/authenticator depending upon several factors such as the frequency of authentication, storage space available at the authenticator, *etc.*

4.3.4.1 Setting the Match Threshold (MT) value

In a practical scenario, the operating conditions during the *authentication phase* could be very different from that of the *enrollment phase*. As a result, the responses generated during authentication could widely differ (beyond the MT value) from the corresponding *golden responses* even after undergoing error correction. This could lead to failed authentication for a genuine *C-PUF* device. Hence, setting the appropriate MT value is an extremely important step in the *C-PUF* design cycle. It should be set such that the resulting HD from the response comparisons is less than or equal to it for the genuine device. Hence, in the proposed design, MT is a function of the operating conditions during authentication, *e.g.,* temperature, aging, *etc.* Moreover, the bit-flip behavior varies substantially across different DRAMs under these operating conditions, as was depicted in Fig. 4.6 for two such modules at different temperatures. Note that, in the *C-PUF* design, it is the DRAM's response that serves as the *final* response and undergoes comparison with the *golden response*. Hence, MT is also a function of the DRAM module present inside *C-PUF*.

Setting the MT value begins with choosing the appropriate DRAM error-correction (strength, specified by $<n,k>$ for HED). Specifically, for a given range of operating conditions, we need to choose the error-correction to be employed at each of the *enrollment points* (E_Ps). To do this, *C-PUF* is first made to go through the *enrollment phase* at an E_P using a set of sample challenges (details in Section 4.3.4.2). This is followed by the *authentication phase* at the corresponding A_Ps. The initial difference (without DRAM error correction), D_{Init}, between the responses during authentication and the corresponding *golden responses* is recorded for each A_P. Next,

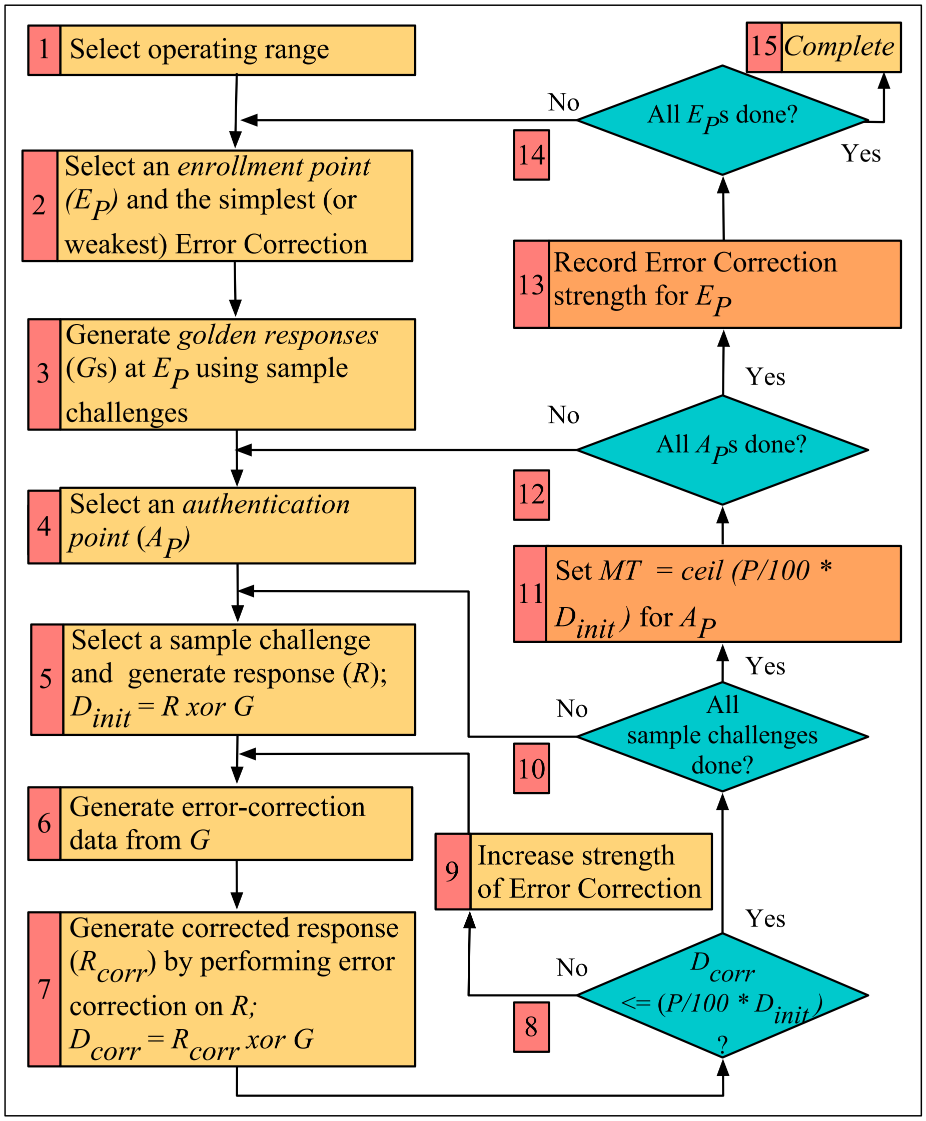

1 Select operating range
2 Select an enrollment point (E_P) and the simplest (or weakest) Error Correction
3 Generate golden responses (Gs) at E_P using sample challenges
4 Select an authentication point (A_P)
5 Select a sample challenge and generate response (R); D_init = R xor G
6 Generate error-correction data from G
7 Generate corrected response (R_corr) by performing error correction on R; D_corr = R_corr xor G
8 D_corr <= (P/100 * D_init) ?
9 Increase strength of Error Correction
10 All sample challenges done?
11 Set MT = ceil (P/100 * D_init) for A_P
12 All A_Ps done?
13 Record Error Correction strength for E_P
14 All E_Ps done?
15 Complete
Yes
No

Fig. 4.9.: Flowchart showing setting of Match Threshold

the error-correction is set such that it is strong enough to bring this difference to less than $P\%$ of D_{Init} for every A_P; P is an user-specified parameter. So, for each A_P,

$$D_{Corr} <= \frac{P}{100} * D_{Init} \tag{4.1}$$

In the current design, the P value was set at 10. Finally, the MT value at the A_P is calculated as,

$$MT = ceil(\frac{P}{100} * D_{Init}) \tag{4.2}$$

ceil() rounds up the value to the nearest higher integer. The whole process is repeated till all the E_Ps are covered. Fig. 4.9 depicts the complete process of setting the MT value in the form of a flowchart. Table. 4.5 and Table. 4.6 (in Section 4.5), list the error-correction (strengths) and MT values for several *C-PUF* instances at different sample A_Ps. As shown for PUF-1, we choose HED with a strength of <31,26> for E_P = 30°C, which caters to A_P = 20°C, 30°C, and 40°C. On the other hand, E_P = 50°C, which caters to A_P = 50°C and 55°C, utilizes HED with a strength of <15,11>. Note that a set of A_Ps could use the same MT value, for example, all A_Ps in the range [20°C, 25°C] could use the MT value corresponding to A_P = 20°C.

Another important parameter in the *C-PUF* authentication methodology is *maximum corrected bits* or B_{Max}, specified inside square brackets in Table. 4.6. B_{Max} sets an upper limit on the maximum number of bits that are corrected in the response generated from the DRAM in the *DRAM Error Correction* stage (step [6] in Fig. 4.4) during authentication. For each A_P,

$$B_{Max} = D_{Init} \tag{4.3}$$

Note that B_{Max} information is contained in the *Error-correction data* field of the challenge, which is sent to *C-PUF* during authentication. This parameter is introduced to protect against a certain kind of attack that, we envision, could be mounted on a *C-PUF* device containing DRAM in the form a removable/replaceable SODIMM. Assuming an attacker has physical access to the device, he/she may be able to replace the DRAM SODIMM with a counterfeit one. During authentication, the *DRAM Error Correction data*, without the knowledge of the underlying counterfeit DRAM, would try to correct the response of the counterfeit DRAM and bring it closer to the *genuine* response. Without an upper bound on the maximum number of corrected bits, the HD between the responses could go below the specified MT value, resulting in successful authentication. To prevent such a scenario, *C-PUF* restricts the error correction to the value specified in B_{Max}. In an actual setting, a *red flag* could be triggered by the device whenever the number of corrected bits reaches B_{Max}, prompting the authenticator to take action, *e.g.,* re-initiate authentication. Note that, similar to MT, B_{Max} takes into account the behavior of the (genuine) DRAM module as well the current operating conditions. A case-study involving the envisioned attack is presented through experiments in Section 4.5.4.

4.3.4.2 DRAM Error Characterization

The previous section presented the methodology for setting the appropriate DRAM error-correction as well as MT, both of which require a good understanding of the DRAM (bit-flip) behavior under different operating conditions. We achieve this by performing *DRAM error characterization*, which was introduced in Chapter 3. It involves an iterative process where the DRAM is subjected to a set of sample challenges under different operating conditions and made to undergo *refresh-pausing* to generate the responses. The sample challenges are constructed by varying the different parameters/factors that affect a DRAM, *e.g., Bitstream_C*, *refresh-pause interval*, *wrapper pattern*, *etc.*. Table 4.1 presents the values of these parameters and operating conditions. An analysis of the generated responses follows, giving us vital insights into the DRAM's behavior. Note that *DRAM error characterization* presents a one-time overhead only since it needs to be performed once by the designer. Apart from these benefits, it also helps to identify the minimum *refresh-pause intervals* for a DRAM module (or a particular block) that meet the entropy requirements. This, in turn, helps to ensure low operational latency in the design.

Next, we present a design feature of *C-PUF* that imparts substantial protection against security attacks.

4.3.5 Intrinsic Reconfigurability

Reconfigurability refers to the ability of a PUF to undergo reconfiguration, *i.e.,* modify its challenge-response behavior. The new behavior, which is unpredictable and cannot be modeled based on the knowledge of the behavior prior to reconfiguration, can substantially enhance the PUF's ability to resist various attacks [14, 71, 73].

Table 4.1.: Parameter values and operating conditions varied during *DRAM Error Characterization*

Addr__D	**Size__D**	**Bitstream__C**	**Wrapper patt.**	**Refresh-pause int.**	**Temperature**	**Aging**
Varied across address space	32 KB, 64 KB, 128 KB	All ‘1’s, all ‘0’s, checkered (101...)	All ‘1’s, all ‘0’s, checkered (101...)	20s, 40s, 60s	20°C, 30°C, 40°C, 50°C, 55°C	6 months, 12 months

Table 4.2.: (SRAM, DRAM) used in different *C-PUF* instances

PUF-1	**PUF-2**	**PUF-3**	**PUF-4**	**PUF-5**
ISSI, DSL	ISSI, DSL	TI, Hynix	TI, DSL	TI, Samsung

Table 4.3.: Parameter values used in the experiments

Fig.	**Addr__S**	**Size__S**	**Addr__D**	**Size__D**	**Bitstream__C**	**Wrapper patt.**	**Refresh-pause int.**
4.11, 4.12	Ten diff. values	32 B	Ten diff. values	128 KB	One (fixed) value	All ‘1’s	40s
4.13, 4.14	0x2500	32 B	Ten diff. values	128 KB	Ten diff. values	All ‘1’s, all ‘0’s, checkered (101...)	40s
4.15, 4.16	0x2500	32 B	Five diff. values	128 KB	Two diff. values	All ‘1’s	40s
4.17	0x2500, 0x4000	32 B	0x38000000	128 KB	One (fixed) value	All ‘1’s	20s, 40s, 60s

The proposed design achieves reconfigurability in *C-PUF intrinsically*, *i.e.,* without using any additional resource, unlike [54]. It specifies two modes of reconfiguration in *C-PUF* – SRAM reconfiguration and DRAM reconfiguration. SRAM reconfiguration involves changing *Addr_S* in the challenge-response mechanism, generating (new) start-up values from a different block in the SRAM. DRAM reconfiguration, on the other hand, modifies the *Refresh-pause interval*, generating new bit-flip patterns from the same DRAM block. Hence, by entering one or both reconfiguration modes, *C-PUF* can undergo reconfiguration *intrinsically* and start behaving as a new PUF, as shown through experimental results in Section 4.5.5. Note that (unlike other parameters) the reconfiguration knobs, *Addr_S* and *Refresh-pause interval*, need not vary across different challenges (or authentication runs) except when there is a need for reconfiguration. Also, since the SRAM address space is usually small and could be easily exhausted, we envision that *C-PUF* would undergo more DRAM reconfigurations than SRAM reconfigurations, as depicted in Section 4.5.5.

4.4 Experimental Setup

This section provides a brief description of the experimental setup used to validate the *C-PUF* design. It consists of two Terasic TR4-230 development boards [65], each containing an Altera Stratix IV GX FPGA, 2 MB SRAM, and 1 GB DDR3 DRAM SODIMM. The SRAM is soldered into the boards and cannot be replaced, unlike the DRAM. Hence, to emulate multiple (five) *C-PUF* instances, we also used the 16 KB SRAMs in three TI MSP4305438A micro-controllers. Altogether, a total of five *C-PUF* instances were constructed for validation, each consisting of an SRAM and a DDR3 DRAM SODIMM. As shown in Table 4.2, the SRAMs belonged to two different manufactures while the DRAMs were procured from three different manufacturers. Experiments pertaining to varying operating conditions *viz.* temperature and aging, were performed by operating the TR4-230 development boards inside the Quincy Lab 12-140E Incubator. Fig. 4.10 shows the complete experimental setup.

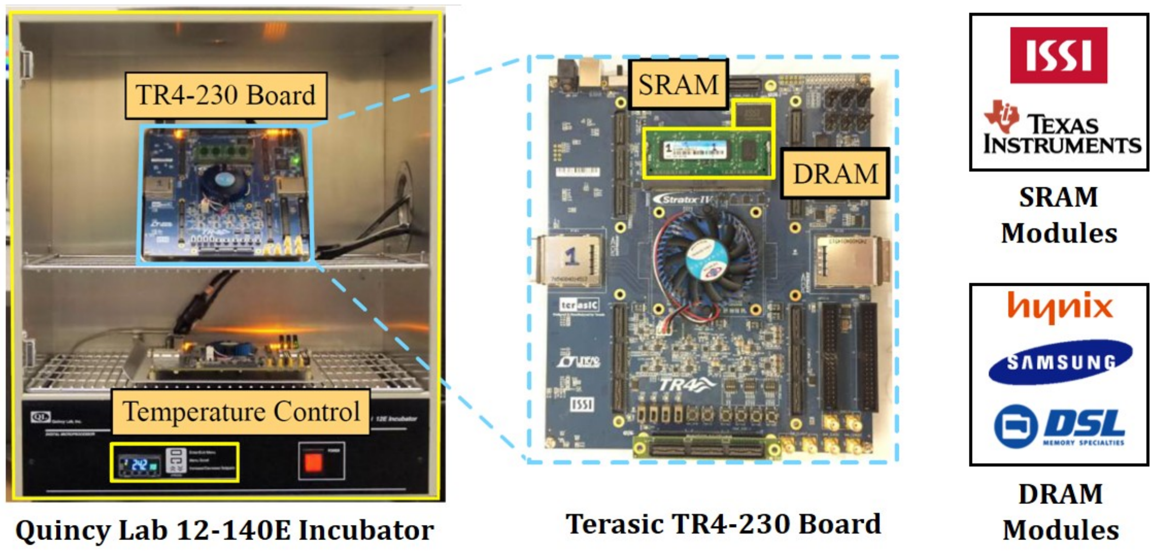


Fig. 4.10.: Experimental Setup

The FPGA was programmed with a soft Nios II processor [66] along with an Altera UniPHY DDR3 memory controller for controlling the DRAM. A custom slave running on the processor was also created, which can instruct the memory controller to pause the DRAM refresh operations. To prevent interference to other running applications (on the device), one could utilize the Partial Array Self-Refresh functionality available in LPDDR DRAMs, which allows selective modification (increase) of the refresh-pause interval for a portion of the DRAM (used by *C-PUF*) [67]. Alternatively, techniques such as *selective DRAM refresh* and *memory ballooning* [38] could also be utilized. Similarly, for *PUF-1 and PUF-2* that use the SRAM on the TR-230 board, the start-up values were generated by *power-cycling* the whole board. Alternatively, one could either adopt the mechanism described in [45] that does not require *power-cycling* to generate start-up values or utilize power-gating techniques targeting only the SRAM. For the remaining PUFs, *PUF-3* – *PUF-5*, the SRAM start-up values were generated on a different platform and subsequently transferred to the TR-230 board in real-time. Also, in these proof-of-concept implementations, the operations in the *SRAM/DRAM Error Correction*, *XOR*, and the *HASH* stages were carried out using software (or firmware) executing on the Nios II processor. Alternatively, dedicated on-chip hardware such as a cryptographic engine, ECC for memories, *etc.,* could be utilized either as already-available units or by adding them to the system. Note that the generic architecture of *C-PUF*, described in this chapter, is independent of any specific implementation.

4.4.1 Implementation Overhead and Performance

The *C-PUF* instances, described above, utilized off-the-shelf (memory) modules already present in an embedded system. Moreover, the various operations – *SRAM/DRAM Error Correction*, *XOR*, and the *HASH* stages were carried out using software (or firmware) executing on the Nios II processor. Hence, there is no additional HW overhead in the current implementation. However, there is some latency overhead, which we present in the form of *authentication response time*. The latter, which could be used as a performance metric, represents the time taken by *C-PUF* to generate an authentication decision, *i.e.,* the time interval between the application of a challenge and the generation of the (final) response during an *authentication phase* (Section 4.3.4). There are three primary contributors to the *authentication response time*. The first one is SRAM *power-cycling*, which is used to generate the startup values. This process is very fast and takes only a few milliseconds. The second one comprises the various operations performed using software, *viz. SRAM/DRAM Error Correction*, *XOR*, and the *HASH*. The total execution time for these is in the order of a few hundred milliseconds. The third contributor, which most affects the *authentication response time*, is DRAM refresh-pausing. This process is relatively much slower (∼seconds), as shown through the *refresh-pause interval* parameter values in Table. 4.3. Therefore, in the current implementation, the *authentication response time* is primarily the *refresh-pause interval* (value) used in the challenge. As described in Section 4.3.4.2, *DRAM error characterization* can identify the minimum *refresh-pause* *intervals* for a DRAM module (or block) that meet the entropy requirements. This could help to ensure low *authentication response time* in the design.

4.5 Results

This section presents the results obtained from experiments conducted to validate our work. It is broadly divided into five subsections. In the first two, we analyze the *C-PUF* design in the light of *entropy* and *number of CRPs*, while also comparing it with SRAM and DRAM PUFs. Next, the robustness of the design is showcased through exhaustive authentication tests pertaining to wide temperature variations and aging effects. Thereafter, we emulate an invasive attack on *C-PUF* and analyze its ability to resist the same. Finally, we present a sample timeline of *C-PUF's* operation to highlight *intrinsic reconfigurability* in its design. Table 4.3 provides a summary of the parameter values (in challenges) used in the different experiments.

4.5.1 Entropy Analysis

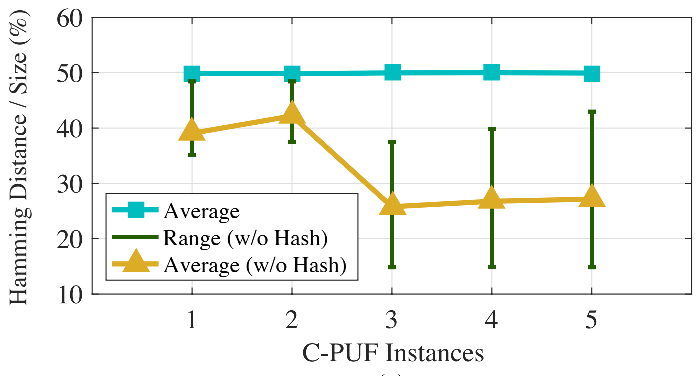
As described earlier (Section 4.2), *entropy* manifests itself in the from of *uniqueness* and *unpredictability.* Hence, we analyze *C-PUF* in the light of these two qualities and compare it with an SRAM PUF and a DRAM PUF.

4.5.1.1 Uniqueness Analysis

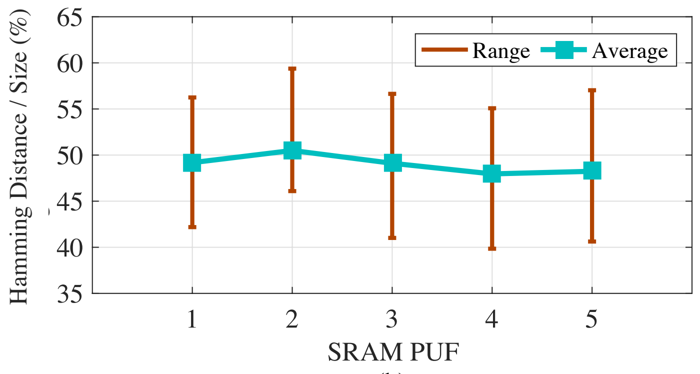
The ability of a PUF in generating unique responses (or fingerprints) forms the very foundation of challenge-response-based authentication. In the present work, we quantify *uniqueness* for a PUF as the percentage of bits that are different in the PUF's response when compared to another (same-type) PUF's response.

To demonstrate the uniqueness of the responses generated by the proposed design, five *C-PUF* instances (PUF-1 – PUF-5) were each subjected to ten different challenges at 20°C. Table 4.3 provides a summary of the parameter values used in the challenges. Note that, unlike during *DRAM error characterization*, the *Bitstream_C* field in the challenges consisted of a random sequence of bits, generated using a pseudo-random number generator. Next, for every challenge, the responses (128 KB) generated by one instance was compared against the ones generated by every other instance. Fig. 4.11(a) shows the range and average uniqueness resulting from these comparisons for each of the instances. As described in Section 4.3, *C-PUF* generates the responses by *xor-ing* (in *XOR* stage) and *hash-ing* (in *HASH* stage) the start-up value of the SRAM and subsequently applying it to the DRAM. Hence, one may argue that the uniqueness of the responses generated by *C-PUF* is contributed by the *hash* (SHA-256) operation only and the not by the rest of the design. Hence, we generated the responses both with and without the *hash* operation, as depicted in Fig. 4.11(a). Note that, with the *hash* operation, response generation follows the generic flow described earlier in Section 4.3. As shown, a high average uniqueness value of 50% (524,288 bits) was obtained for each of the instances, which shows that the instances truly exhibit unique behavior. Due to limited spread of the minimum and maximum values, the range is not shown for this case. Next, without the *hash* operation, the responses were generated by skipping the *HASH* stage altogether, *i.e.,* by directly applying the *xor-ed* start-up value of the SRAM to the DRAM; the resultant comparisons are also shown in Fig. 4.11(a). The average value of 25% (337,600 bits) across the instances highlights the uniqueness inherently provided by the *C-PUF* design. This uniqueness can be visually perceived through Fig. 4.12 that gives a pictorial representation of the responses generated by the *C-PUF* instances (with the *hash* operation) when subjected to the same (one) challenge; each response is a fingerprint of the device containing the instance.

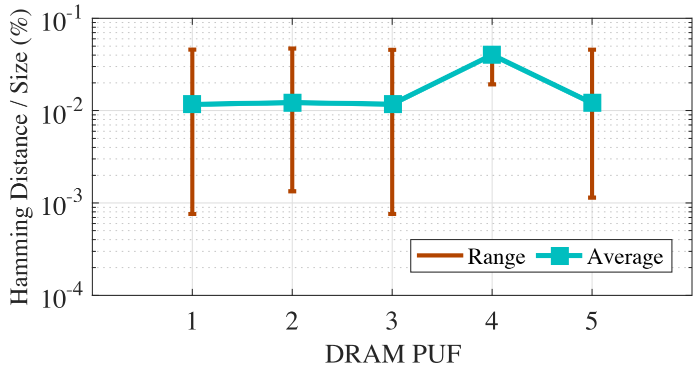
To further showcase the enhanced uniqueness provided by *C-PUF*, we also compared it with that of an SRAM PUF and DRAM PUF. Using the same memory modules (present in the *C-PUF* instances), five SRAM PUFs and DRAM PUFs (each) were constructed. Next, these were subjected to the same ten challenges (at 20°C), followed by a similar comparison of the generated responses. Note that the DRAM-



(a)



(b)



(c)

Fig. 4.11.: Uniqueness Analysis for ten different challenges

specific parameters in the challenges were ignored when generating responses from an SRAM PUF and *vice-versa*. Also, the response size for an SRAM PUF and DRAM PUF was 32 B and 128 KB, respectively, which is the same as was generated by the respective memory modules in a *C-PUF* instance. As described earlier, SRAM PUFs exhibit high uniqueness, which is evident in Fig. 4.11(b) with an average unique-

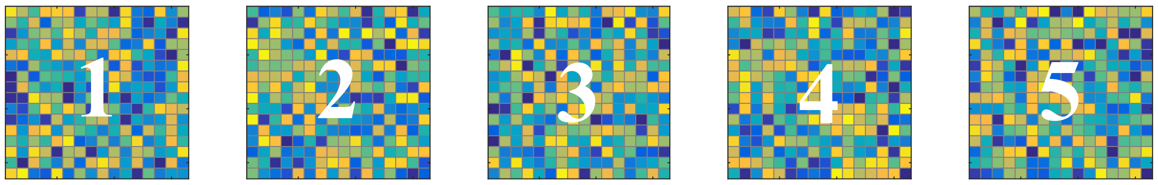


Fig. 4.12.: Responses (fingerprints) from different *C-PUF* instances

ness value of 49%. On the contrary, the uniqueness exhibited by DRAM PUFs is comparatively much lower (by more than three orders of magnitude), as depicted in Fig. 4.11(c). Hence, for uniqueness, *C-PUF* closely resembles an SRAM PUF but substantially improves over a DRAM PUF.

4.5.1.2 Unpredictability Analysis

In the present work, we quantify *unpredictability* as the probability of incorrectly predicting a PUF's response to an unknown challenge with the knowledge of a certain number of (previously-used) CRPs. Ref. [74] captured this notion by investigating the extent of change in the responses when the corresponding challenges were varied by a certain number of bits. We followed a similar methodology and changed the challenges by just one bit. The change observed in the corresponding responses gave us an estimate of the unpredictability in *C-PUF*. However, as depicted in Fig. 7, *C-PUF's* challenge has several variable parameters. The unpredictability analysis was performed for each of these parameters, however, due to lack of space, we discuss and present the results for three of these – *Addr_S*, *Addr_D*, and *Bitstream_C*. Table. 4.4 presents the results of the analysis.

Addr_S specifies the (starting) address of the SRAM block that undergoes power-cycling to generate the start-up values. Five addresses or blocks were (randomly) selected and used to construct five *master challenges*. Each *master challenge* has one of the selected addresses as *Addr_S*; the rest of the parameters (Table. 4.4) were chosen randomly but were fixed across all the master challenges. Next, we generated

Table 4.4.: $\text{HD}_{\text{Response}}$ values for different parameters[1]

Parameter	**Average**	**Maximum**	**Minimum**	**Median**
Addr_S	524,244	525,113	523,373	524,294
Addr_D	68	84	55	68
Bitstream_C	167	301	118	147

the set of all possible challenges (*derived challenges*) that differ in just one bit from the master challenge in the *Addr_S* parameter (*i.e.,* Hamming Distance, HD = 1). Thus, five different sets of derived challenges were generated, each corresponding to a master challenge. Next, *C-PUF* was subjected to these challenges at 20°C, generating five master responses and five sets of derived responses. In each set, the derived responses were then compared with the master response to generate the $\text{HD}_{\text{Response}}$. As shown in Table. 4.4, changing just one-bit in the challenge changed the response by 524,244 bits (average). Thus, even with the knowledge of a certain number of CRPs, it is near impossible for an attacker to predict the response to a challenge that differs in even one bit from the CRPs.

Addr_D specifies the (starting) address of the DRAM block that undergoes refresh-pausing to generate the bit-flip patterns. Unpredictability analysis was performed by following a similar methodology as with *Addr_S* (described above). However, in this case, five random DRAM addresses or blocks were used to construct the master challenges, and the derived challenges were generated such that they differ from the master challenge in the *Addr_D* parameter in just one bit. As shown in Table. 4.4, changing just one bit in the challenge changed the response by 68 bits (average). Thus, the unpredictability associated with *Addr_D* is comparatively lower than *Addr_S* and is indicative of the low entropy/uniqueness of a (refresh-pausing-based) DRAM PUF. However, even with just a one-bit difference in the challenge, the attacker still needs

[1]Temp. = 20°C; *Size_S* = 32 B, *Size_D* = 128 KB, *Wrapper pat.* = All '1's, *Ref.-pause int.* = 40s

to guess the value and position of (an average) 68 new bits (out of 128KB), of which there are ${}^{128\times1024\times8}C_{68} \times 2^{68}$ different possibilities. Moreover, an increase in operating temperature leads to an exponential change and would only make the attacker's job more difficult. Hence, *C-PUF* offers high unpredictability with respect to *Addr_D* too.

Bitstream_C is a random binary sequence that, after undergoing *xor* and *hash* with the SRAM start-up values, is written onto the DRAM for refresh-pausing. Following a similar methodology as with the previous two parameters, five random bitstreams were generated to construct five master challenges. The derived challenges were then generated such that they differ from the master challenge in the *Bitstream_C* parameter in just one bit. As shown in Table. 4.4, changing just one bit in the challenge changed the response by 167 bits (average). As with *Addr_D*, the unpredictability associated with *Bitstream_C* is comparatively lower than *Addr_S* and is indicative of the low entropy/uniqueness of a (refresh-pausing-based) DRAM PUF. However, even with just a one-bit difference in the challenge, the attacker still needs to guess the value and position of (an average) 167 new bits (out of 128KB), of which there are ${}^{128\times1024\times8}C_{167} \times 2^{167}$ different possibilities. As before, an increase in operating temperature leads to an exponential change and would only make the attacker's job more difficult. Hence, *C-PUF* also offers high unpredictability with respect to *Bitstream_C*.

4.5.2 CRP Analysis

C-PUF supports an exponential number of CRPs that makes it suitable for challenge-response-based authentication. To better explain this, we first estimate the number of CRPs supported by an SRAM PUF and a DRAM PUF. We assume the same memory configuration as used in the experiments – 2 MB SRAM and 1 GB DRAM, generating 32 B and 128 KB responses, respectively. Accordingly, the number of CRPs supported by the SRAM PUF is N_S, where $N_S = 2^{16}$ and represents the number of 32 B blocks in a 2 MB address-space. Unlike an SRAM PUF, the challenge applied to a DRAM PUF (block) consists of several widely-variable parameters, as explained in Section 4.3.1. One such parameter is *Bitstream_C*, which is a random binary sequence of *Size_D*. Note that *Size_D* also specifies the size of the DRAM block (128 KB, here). Hence, the total number of ways *Bitstream_C* could be varied is of the order of $2^{128\times1024\times8}$. With a 1 GB DRAM module (containing 2^{13} blocks) and three possible values for Wrapper patterns and Refresh-pause intervals each, the total CRPs supported by the DRAM PUF equals N_D, where $N_D = 2^{128\times1024\times8} \times 2^{13} \times 3 \times 3$. As described earlier, *C-PUF* tightly integrates the response generation mechanisms of an SRAM PUF and a DRAM PUF, and hence it supports a total of N_C CRPs, where $N_C = N_S \times N_D$ or $2^{1,048,608}$. Thus, it supports an exponential number of CRPs, which is ideal for challenge-response-based authentication.

4.5.3 Robustness Analysis: Authentication under Varying Operating Conditions

Robustness of a system can be broadly defined as its ability to withstand or function normally under varying environmental and temporal conditions. In the present context of PUF-based authentication, it refers to the PUF's ability to undergo successful authentication under different operating conditions, primarily determined by *temperature* and *aging*. We now present a robustness analysis of the proposed *C-PUF* design.

4.5.3.1 Authentication under Temperature Variations

As shown earlier in Figs. 4.5 and 4.6, variations in operating temperatures can strongly affect the PUF's responses, thereby hindering its ability to perform authentication. However, through the proposed *C-PUF* design and the associated authentication scheme, we address this challenge successfully. We demonstrate this for a sample operating range of [20°C, 55°C]. Note that the design is not restricted to this

Table 4.5.: Hamming Encoder-Decoder $< n, k >$ values for different *enrollment points*

PUF	**30°C**	**50°C**
1	<31,26>	<15,11>
2	<31,26>	<31,26>
3	<31,26>	<15,11>
4	<31,26>	<31,26>
5	<31,26>	<31,26>

particular range and could easily scale to accommodate wider ranges. As the first step, 100 different challenges were applied to each of the *C-PUF* instances at two different *enrollment points* (E_Ps) – 30°C and 50°C. The generated responses served as the *golden responses* and were stored in the CRP database. Next, to emulate an actual scenario, authentication was performed at five different *authentication points* (A_Ps) – 20°C, 30°C, 40°C, 50°C, and 55°C, by reapplying the same challenges to the instances. Note that, $E_P = 30$°C and $E_P = 50$°C cater to the lower three and upper two A_Ps, respectively. Table. 4.5 lists the error-correction utilized for each of the *C-PUF* instances at different E_Ps. Next, at each A_P, the responses of the *C-PUF* instances generated during authentication were compared with their respective *golden responses* to calculate the *intra-puf* comparison HD, as shown in Fig. 4.13. The *y-axis* represents *relative frequency*, *i.e.,* the fraction of the total comparisons, either *intra-puf* or *inter-puf* (explained later), that yields a certain HD, represented on the *x-axis*. Table 4.6 gives the MT values, which were used to determine the authentication outcome, for each of the instances at different A_Ps. The numbers in square brackets represent the *maximum corrected bits*, B_{Max}, which was used to restrict the error correction during authentication. Note that the appropriate error-correction as well as MT values, used throughout the experiments, were set using the methodology described earlier in Secs. 4.3.4.1 and 4.3.4.2. Overall, the experiments resulted in 2438 successful authentications (out of $2,500$), *i.e.,* a *true-positive* rate of 97.5%.

Table 4.6.: *Match Threshold* and [*Maximum Corrected Bits*] values at different *authentication points*

PUF	**20°C**	**30°C**	**40°C**	**50°C**	**55°C**
1	14 [132]	2 [15]	52 [520]	14 [135]	270 [2700]
2	4 [40]	1 [5]	11 [110]	3 [30]	30 [300]
3	12 [120]	2 [13]	43 [430]	10 [100]	200 [2000]
4	6 [52]	1 [5]	9 [84]	8 [80]	94 [940]
5	10 [94]	1 [8]	44 [440]	4 [40]	50 [500]

The *C-PUF* design also avoids any *false-positives*; an authentication outcome is termed as a *false-positive* when a counterfeit device is successfully authenticated as a genuine one. In the current design, this could happen when the response generated during authentication by a counterfeit *C-PUF* instance (say PUF-X) is very close to the *golden response* of a genuine instance (say PUF-Y), resulting in PUF-X getting falsely authenticated as PUF-Y. To emulate this scenario, we performed *inter-puf* comparisons, *i.e.,* the *golden responses* of a *C-PUF* instance were compared with the responses generated during authentication of every other instance. Fig. 4.13 shows the *relative frequency* versus HD for the *inter-puf* comparisons corresponding to the same 100 challenges and operating conditions. Altogether, a total of 10,000 *inter-puf* comparisons were carried out. At each A_P, the HD resulting from *inter-puf* comparisons was multiple orders of magnitude higher than the MT value (corresponding to the genuine instance), thus ensuring the absence of any false-positives.

4.5.3.2 Authentication under Aging Effects

PUFs, like all electronic components, undergo structural degradation over prolonged use, severely affecting their reliability. Hence, robustness to temporal variations (aging effects) is a much-desired attribute in any PUF design. To show that *C-PUF* is robust to aging affects, we first applied 100 different challenges to two of

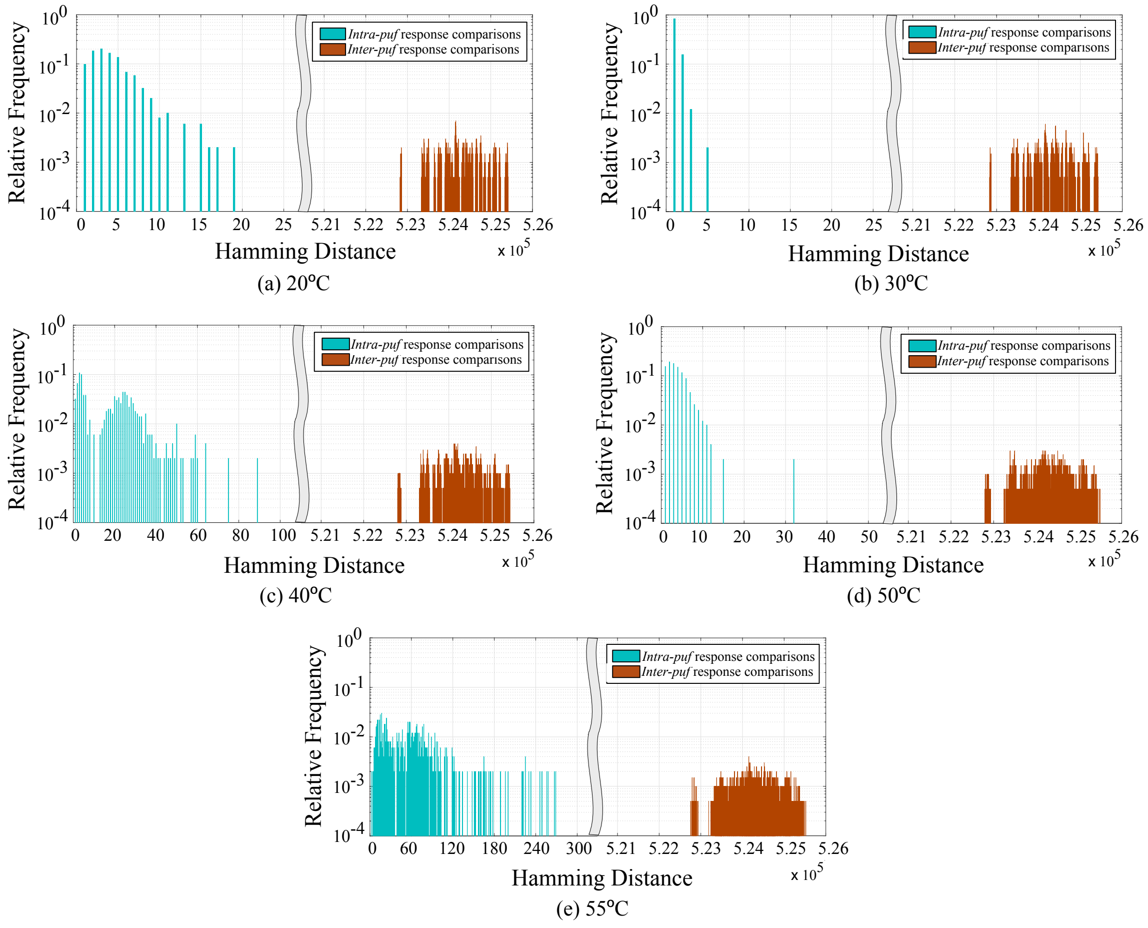


Fig. 4.13.: Authentication under temperature variations

the *C-PUF* instances (PUF-1 and PUF-2) at 30°C, and recorded the corresponding *golden responses*. Next, the instances underwent an accelerated aging process [75] through the application of thermal stress. Specifically, a temperature of 85°C was applied to the instances for 460 hours. To avoid damage to the TR-230 boards from prolonged exposure to high temperature, the aging process was carried out in parts, *i.e.*, the high temperature was applied for 12 hours followed by a cooling time of 4 hours. The Arrhenius equation, typically used for predicting reliability/failure rates,

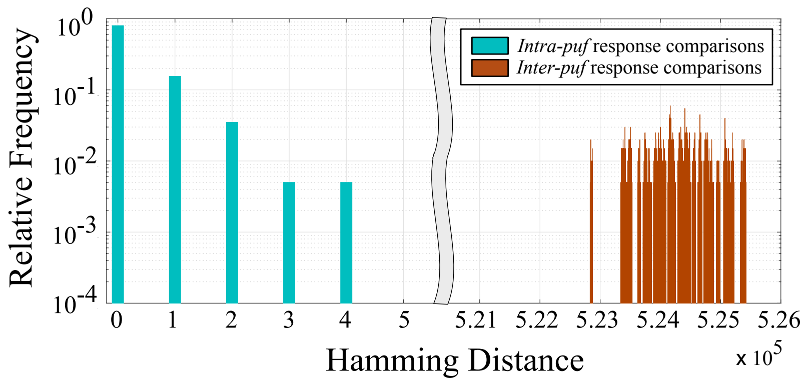


Fig. 4.14.: Authentication under aging effects

was used to calculate the acceleration factor (AF) resulting from the applied thermal stress [68]. As per the Arrhenius equation,

$$AF = e^{\left(\frac{E_a}{k}\left(\frac{1}{T_{\text{use}}} - \frac{1}{T_{\text{stress}}}\right)\right)} \tag{4.4}$$

where,

AF = Acceleration Factor

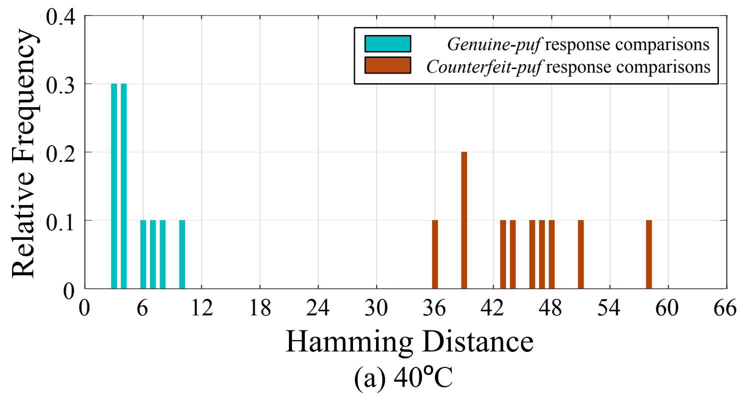
E_a = Thermal Activation Energy (value used = 0.5 eV [75])

k = Boltzmann's Constant (8.63 x 10^{-5} eV/K)

T_{use} = Use Temperature (value used = 303 K (30°C))

T_{stress} = Stress Temperature (value used = 358 K (85°C))

Using these values, AF was calculated to be ~19. Since the thermal stress was applied for 460 hours, this translates to 8740 (19 × 460) hours or 12 months of natural aging at 30°C. The responses were then generated from the aged instances during authentication (with the same challenges and at 30°C) and compared with the golden ones. Fig. 4.14 shows the *relative frequency* versus HD for both *intra-puf* and *inter-puf* comparisons. Generally, aging has a much lesser effect on the response-generation process as compared to temperature. In other words, the responses are primarily influenced by temperature and, hence, we set the MT values as per the 30°C *enrollment*



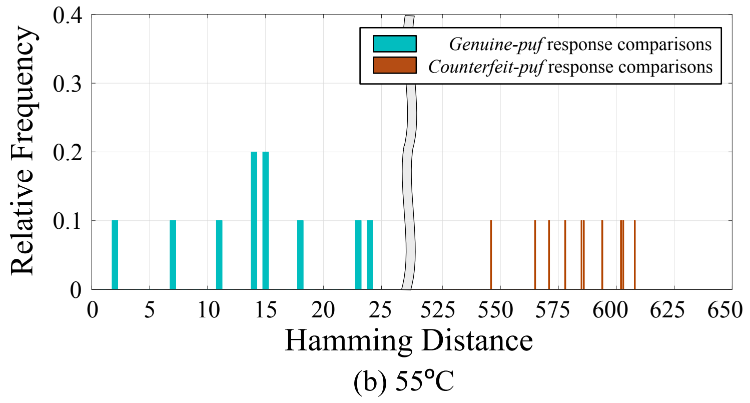
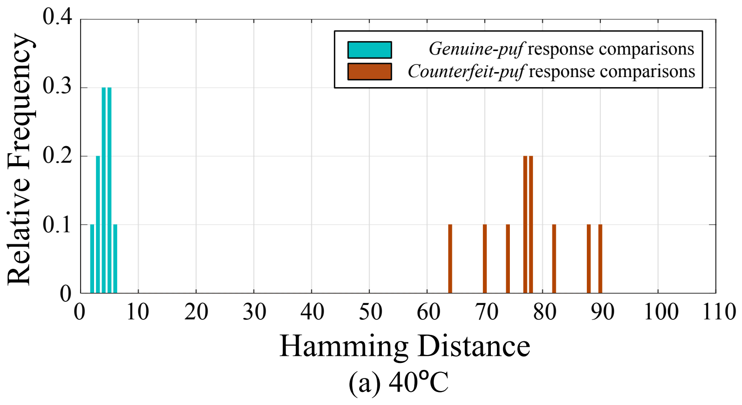


Fig. 4.15.: Authentication of PUF-2C versus PUF-2

point in Table 4.6. This resulted in 198 successful authentications (out of 200), *i.e.*, a 99% true-positive rate, and without any false-positives.

4.5.4 Replacing Genuine DRAM with a Counterfeit

Several embedded systems contain DRAMs in the form of Dual In-line Memory Modules (DIMMs) that (unlike SRAMs, which are physically soldered) could be easily detached from the system. An attacker with physical access to the system may be able to replace the genuine DIMMs with counterfeit ones, and then try to undergo authentication. This scenario was emulated using two *C-PUF* instances, PUF-2 and PUF-4, which represented the genuine systems. First, PUF-2 was made to undergo enrollment at two different E_Ps – 30°C and 50°C, and the corresponding *golden re-*



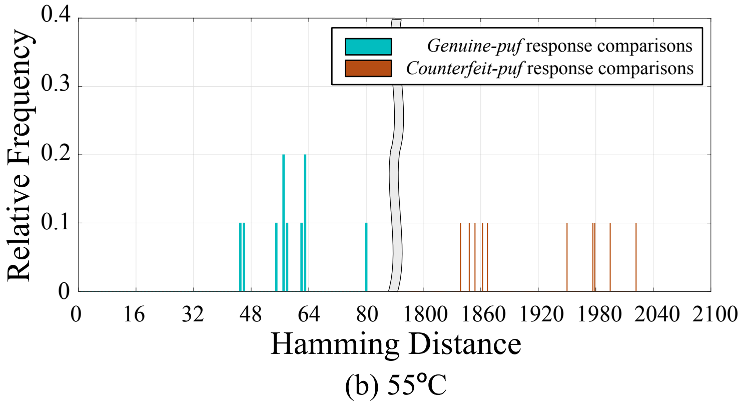


Fig. 4.16.: Authentication of PUF-4C versus PUF-4

sponses were recorded; a set of ten different challenges was used in this process. This was followed by authentication at two different A_Ps – 40°C and 55°C, which were catered to by E_P = 30°C and 50°C, respectively. The responses generated during authentication were then compared with the corresponding *golden responses*; these comparisons are termed as *genuine-puf* response comparisons. Next, a counterfeit PUF, PUF-2C, was built by replacing the DRAM in PUF-2 with the one in PUF-4. Hence, PUF-2C now had the same SRAM as PUF-2 but contained a different (counterfeit) DRAM. Note that the DRAM in PUF-4 was chosen as the replacement in order to emulate the worst-case scenario; the DRAMs (originally) in PUF-2 and PUF-4 have the same make and architecture. The same set of challenges were then applied to PUF-2C, followed by a comparison of the generated responses and the corresponding *golden responses* (of PUF-2); these comparisons are termed as *counterfeit-puf* response comparisons. Fig. 4.15 presents the *relative frequency* versus HD for both *genuine-puf* and *counterfeit-puf* response comparisons. To determine the authentication outcome, we used the MT values corresponding to PUF-2, as specified in Table 4.6. This resulted in unsuccessful authentication for each of the challenges and A_Ps; in other words, the *C-PUF* design was able to detect the counterfeit PUF and prevent any false-positives. To further emphasize this ability, we built another counterfeit PUF, PUF-4C, which had the same SRAM as PUF-4 but contained the DRAM from PUF-2. The above-described process was repeated, and the resulting comparisons are depicted in Fig. 4.16. As before, the counterfeit PUF was detected in each case and no false-positives were generated.

4.5.5 Intrinsic Reconfigurability: A Sample Timeline

The ability to be *intrinsically* reconfigured, *i.e.,* modify its challenge-response behavior, can provide *C-PUF* substantial protection against various attacks (Section 4.3.5). As described earlier, there are two reconfiguration knobs – *Addr_S* (SRAM reconfiguration) and *Refresh-pause interval* (DRAM reconfiguration). To demonstrate that *C-PUF's* behavior does undergo substantial modification, we reconfigured a *C-PUF* instance several times and applied the same challenge to it after each reconfiguration. This is depicted in a sample timeline (T_1–T_8) in Fig. 4.17, where the instance undergoes a series of DRAM and SRAM reconfigurations. The temperature during the whole process was maintained at 20°C. The difference (in terms of HD) between the response generated after a particular reconfiguration and the one generated after the last SRAM reconfiguration is represented by the *x-axis* in Fig. 4.17. For example, the response generated after the DRAM reconfiguration at T_5 differs from the one generated after the SRAM reconfiguration at T_3 by more than 600 bits. This difference increases to more than $500,000$ bits for the two SRAM reconfigurations at T_3 and T_6.

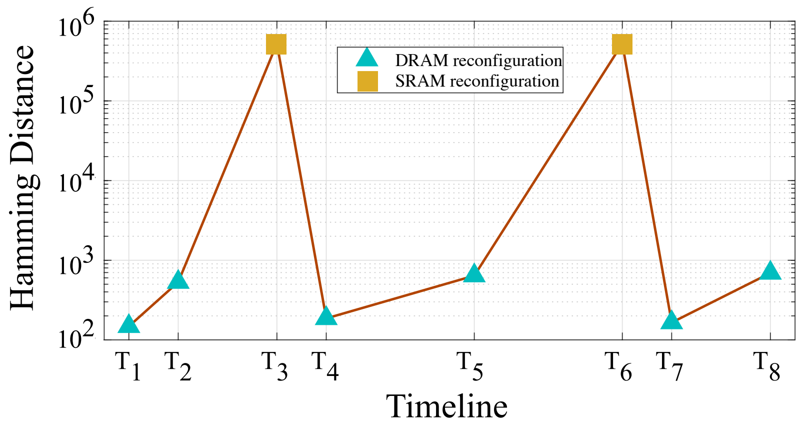


Fig. 4.17.: Reconfiguration timeline

Although not depicted in Fig. 4.17, the choice of the *Refresh-pause intervals* ensures that there is also a substantial difference (that meet entropy requirements) between the responses generated after consecutive DRAM reconfigurations (*e.g.*, at T_4 and T_5). Note that the availability of two reconfiguration modes in *C-PUF* gives it the flexibility to choose one or the other depending on the scenario, as explained in Section 4.6.2.

4.6 Discussions

This section provides a brief discussion of potential attack scenarios as well as additional design aspects of *C-PUF*.

4.6.1 Attack Scenarios

We envision two types of attacks on a device (containing *C-PUF*) – non-invasive and invasive.

In the current design, the communication between the authenticator and the device (containing *C-PUF*) is unencrypted. Hence, we assume that an attacker can eavesdrop on this communication and extract some of *C-PUF's* CRPs used during authentication. This could enable the attacker to mount a non-invasive attack [14], which exploits a large set of (previously-used) CRPs for generating an approximate model of a PUF's (challenge-response) behavior that could predict the (future) responses. However, successfully mounting such an attack on *C-PUF* is extremely difficult, as described through the predictability analysis in Section 4.5.1.2. Specifically, the unpredictable behavior resulting from the coupling (using *hash* and *xor*) of two heterogeneous memory technologies, each with its distinct behavior, leads to an extremely-complex overall model in *C-PUF*. This complexity is further enhanced by the presence of multiple variable parameters in its challenge-response mechanism, which generates an exponential number of CRPs, as shown in Section 4.5.2. Lastly, *C-PUF's* final line of defense against such as an attack lies in its ability to undergo reconfiguration, which completely modifies its challenge-response behavior. For example, *C-PUF* can be reconfigured after every *authentication phase*, cycling among the different reconfiguration points in a random or round-robin fashion. In other words, the reconfiguration knobs can be used just like any parameter in the challenge. Hence, to an external attacker trying to model *C-PUF's* behavior, the responses would seem like coming from multiple completely different PUFs, making the attack extremely time and computation intensive. Due to the above reasons, we believe that *C-PUF* can provide high resistance (and reduced vulnerability) to a non-invasive attack such as [14]. Note that a formal analysis of *C-PUF's* vulnerability to such attacks is currently in progress and is planned as a part of future work. Encrypting the communication between the authenticator and device can also help prevent such attacks; however, this is not a requirement for the *C-PUF* design.

An invasive attack [73] involves physical tampering with the target device and requires physical access to it. We described one such scenario in Section 4.5.4, where the genuine DRAM module was replaced with a counterfeit one, followed by several authentication attempts. As shown, *C-PUF* was successfully able to detect this attack and prevent authentication. Mounting such an attack on the SRAM, on the other hand, is extremely difficult; its on-chip location requires the attacker to possess a very high level of expertise and sophisticated resources. Another type of invasive attack could involve memory-line snooping, where the attacker can (non-intrusively) read off the data in off-chip memory lines during authentication. Note that the SRAM start-up values are always hashed before they travel through the off-chip memory lines. Since *hash* (SHA-256, used here) is a *one-way function*, reading these (hashed) values does not reveal any information about the SRAM's behavior. However, the attacker could be able to read the data written to the DRAM as well as its response. Assuming he/she has access to *C-PUF* for a considerable amount of time ($\sim$minutes/hours), he/she may then be able to derive a portion of the DRAM's (bit-flip) behavior. However, entirely constructing this behavior requires a very large number of CRPs, which is extremely time and computation intensive due to the huge (exponential) challenge-response space of the DRAM (Section 4.5.2). In summary, the utilization of heterogeneous memory technologies, which are spatially distributed (on-chip and off-chip), substantially improves *C-PUF's* resistance to such invasive attacks.

4.6.2 C-PUF versus [SRAM PUF + DRAM PUF]

An alternative design (say, *SD*) could have both an SRAM PUF and a DRAM PUF present in a device but operating independently of each other. *C-PUF* scores over such a design by providing much stronger defense against non-invasive attacks owing to its more complex (challenge-response) behavior, as described in the previous subsection. One manifestation of this complexity is the higher number of CRPs that are supported by *C-PUF*, as compared to *SD*. Section 4.5.2 showed this through a typical SRAM/DRAM configuration in *C-PUF*. Specifically, if the number of CRPs supported by the SRAM PUF and DRAM PUF are N_S and N_D, respectively, then *C-PUF* supports a total of $N_D = N_S \times N_D$ CRPs, as described in Section 4.5.2. In contrast, *SD* can support a maximum of N_{SD} CRPs, where $N_{\mathrm{SD}} = N_S + N_D$, which is multiple orders of magnitude lower.Moreover, the availability of two reconfiguration modes in *C-PUF* gives it the flexibility to choose one or the other, depending on the scenario. For example, DRAM reconfiguration could be utilized in case of low available address space in SRAM while SRAM reconfiguration could be performed when fast authentication is required. *SD* does not have the ability to make these decisions as the constituent PUFs operate independently of each other.

4.6.3 Effect of Variations in Supply Voltage

Variations in supply voltage could affect a *PUF's* operation. However, due to the presence of on-board voltage/power regulators (in all embedded systems), which closely monitor power supply to the components, this is an extremely rare (and short-duration) phenomenon. Nevertheless, with regards to *C-PUF*, the SRAM is unaffected by supply-voltage variations as its response-generation mechanism relies on *power-cycling* or an intentional lowering of supply-voltage as in [45]. The DRAM could see additional bit-errors if the variation goes beyond the manufacturer-specified voltage *guardbands* [76, 77], however, the error-rate is low [76] and can be handled by the authentication scheme employed in *C-PUF*. The only operations that could be truly affected by supply-voltage variations are the *HASH*, *XOR*, and SRAM/DRAM *Error-correction*, as they involve arithmetic computations. In such a scenario, its fair to assume that the whole compute sub-system (of the embedded system) is actually paralyzed, affecting all running applications (not only authentication using *C-PUF*). Hence, if authentication is in progress, it will need to be stopped and re-initiated once the voltage levels are restored.

4.7 Conclusion

In this chapter, we propose the concept and design of a memory-based combination PUF that tightly integrates two heterogeneous memory technologies to construct a PUF that overcomes several shortcomings of current memory-based PUFs. The design can be easily implemented on a COTS device and takes a step towards multi-component authentication in an embedded system. The PUF was implemented in a real system using several off-the-shelf SRAMs and DRAMs. Experimental results demonstrated substantial improvements over current memory-based PUFs including the ability to resist various attacks. We also proposed a light-weight authentication scheme that ensures robust operation of the PUF across wide environmental and temporal variations. Extensive authentication tests performed on several PUF prototypes achieved greater than 97.5% true-positive rate across these variations. The absence of any false-positives, even under an invasive attack, further highlighted the effectiveness of the overall design. Inclusion of other (system) components into the *C-PUF* design and formal analysis of its vulnerability to non-invasive attacks (*e.g.,* machine-learning based) are currently in progress and are planned as part of future work.

5. A LIGHTWEIGHT END-TO-END AUTHENTICATION PROTOCOL FOR IMPLANTABLE MEDICAL DEVICES

The past decade has witnessed a rapid growth in the use of Implantable Medical Devices (IMDs) for monitoring and treating a variety of medical conditions, with their global market valuation expected to reach $50 billion by 2024 [17]. IMDs are increasingly being equipped with wireless interfaces [18], allowing them to communicate with an External Device (ED) such as a doctor's programmer or a patient's smartphone, as shown earlier in Fig. 2.4. While this greatly improves standard of care (allowing for post-deployment tuning of therapy as needed and remote, real-time access to health data), it also exposes the IMD to a range of security concerns [3, 4, 19–21] such as potential interaction with untrusted EDs and potential leakage of confidential medical data.

Several techniques [22–30] have been proposed to address these concerns. One such technique [28, 30], as shown earlier in Fig. 2.4, leverages a *trusted entity* such as a Health Server (HS), which assumes the role of an authenticator and arbitrates access to an IMD when an ED requests it. Upon successful authentication of an ED, the HS distributes a shared key to both the IMD and the ED. This key is used to encrypt all further communication between the IMD and ED using symmetric-key cryptography, while the ED and HS employ asymmetric (public-key) cryptography to communicate with each other (since they are not as energy-constrained as the IMD). Note that the trusted-entity approach is popular due to its simplified but secure usage model and requirement of no additional devices. However, there are three unaddressed challenges with this approach. First, due to their severely limited energy budget, IMDs [18] typically utilize short-range communication technologies such as Bluetooth LE and, hence, do not have direct network connectivity with a remote HS. This is in contrast to an ED, such as a smartphone, which can use long-range wireless technologies such as Wi-Fi, LTE, *etc.,* for network access. Second, involving the HS for every single authentication session can incur significant energy overheads at the IMD, besides increasing the load on the HS and the network. This, in turn, gives rise to the third challenge. If the HS is not reachable over the network, the IMD and ED cannot securely communicate with each other. Besides, some of these techniques [28, 29] require the (resource-constrained) IMD to perform asymmetric cryptography operations resulting in high energy overhead.

5.1 Chapter Contributions

In this chapter, we propose a lightweight end-to-end authentication and key-exchange protocol based on the trusted-entity approach that fully addresses each of the aforementioned challenges. A key requirement of the proposed protocol is the availability/generation of unique identifiers/keys and random numbers on demand. Although other techniques such as statically-stored secret keys and pseudo-random number generators could be utilized to satisfy this requirement, we utilize a Physically Unclonable Function (PUF) [33] and present its seamless integration with existing cryptography techniques in the protocol. The result is a robust, secure, and lightweight protocol, which protects against various security attacks [19–21] and can be easily implemented using Commercial-Off-The-Shelf (COTS) devices with minimal or no additional hardware resources. Several PUF-based authentication techniques have been proposed earlier, however, they suffer from several shortcomings such as non-applicability to the unique IMD ecosystem (Fig. 2.4) [31, 32], need for special hardware [29, 30], and high operation overhead [29, 30]. The proposed protocol, on the other hand, overcomes these shortcomings and those of earlier trusted-entity approaches [28]. Specifically, we make the following contributions [10]:

- We propose an end-to-end authentication and key-exchange protocol for an IMD ecosystem that addresses several challenges and limitations associated with prior trusted-entity approaches while protecting against a wide range of security and privacy attacks. The protocol requires no special hardware on the IMD or ED, and therefore, can be implemented on COTS devices.

- We present the seamless integration of existing cryptography techniques with a PUF in the proposed protocol while also addressing the limitations and challenges of prior PUF-based approaches in their application to the IMD ecosystem.
- We implement the proposed design on a real system comprising of an ARM Cortex-M0+ based microcontroller as the IMD, a Raspberry Pi 3 B+ as the ED, and an Intel Xeon E5 processor based HS. We perform an extensive evaluation of the design and present experimental results that demonstrate the effectiveness of the protocol in providing robust, secure, and lightweight authentication and key-exchange in IMDs. With a typical IMD that is powered by a 2 Ah battery at 3.3V, the total energy overhead incurred by the protocol per authentication session is as low as 0.000018% of the IMD's total energy budget.

The rest of the chapter is organized as follows. Section 5.2 describes the motivation behind this work. Next, Section 5.3 presents the design details of the proposed authentication and key-exchange protocol. In Section 5.4, we describe the prototype implementations of the three entities (IMD, ED, and HS) and the experimental setup used to validate the proposed protocol. An analysis of the protocol with regards to overhead and security is also presented in this section. Section 5.5 presents some additional design aspects, results, and discussions associated with the proposed protocol. Finally, Section 5.6 concludes the chapter.

5.2 Motivation

Chapter 2 described several approaches [22–30] towards enabling authentication and key exchange in IMDs. While each of them has its advantages and disadvantages [20, 22], this dissertation utilizes the trusted-entity approach [28, 30] due to its better security capabilities, simplified usage model (for the patient), and the potential

Table 5.1.: Capabilities of the different entities in the IMD ecosystem

	IMD	**ED**	**HS**
Energy Budget	Low (few Ah)	Unlimited [1]	Unlimited
Storage Capacity	Low (few KB)	High (few GB)	Unlimited
Communication	Wireless, Short Range	Wireless/Wired, Short/Long Range	Wired, Long Range
Cryptography Computation	Symmetric	Symmetric/ Asymmetric	Symmetric/ Asymmetric

for seamless integration with COTS devices. As shown in Fig. 2.4, an IMD ecosystem based on the trusted-entity approach comprises of the three (types of) entities: Implantable Medical Device (IMD), External Device (ED), and Health Server (HS). The role and function of each of these entities was described earlier in Chapter 2. Table 5.1, here, lists the typical capabilities of the entities. We now describe the key challenges associated with this IMD ecosystem that form the primary motivation behind the design of the proposed protocol.

As shown in Fig. 2.4, ED communicates with IMD to send configuration or programming data as well as receive the patient's physiological data for monitoring and therapy. There are two important requirements associated with this communication process. First, it must be ensured that ED, which is communicating with IMD, is a legitimate or authentic device. In the absence of this, an attacker can use his own device (ED) to access IMD, compromising the safety and privacy of the patient. Prior work [28, 30] have utilized the trusted-entity approach towards addressing this *i.e.,* leveraging a trusted entity - HS, which assumes the role of an authenticator and arbitrates access to IMD when ED requests it. But, there are several unaddressed challenges associated with this approach, as described in the next section. Moreover,

[1]Powered through a chargeable battery or an AC wall outlet.

Ref. [28], which proposes a *hybrid security protocol* for authentication, assumes an asymmetric crypto-system on the IMD, thereby incurring high energy overhead.

In recent years, PUFs have been used to implement several IMD-specific [29,30] as well as generic (not IMD-specific) authentication protocols [31,32]. Ref. [29] proposed utilizing two PUFs, one in an intra-body IC (IMD) and one on an FPGA (ED), both of which are *matched* to produce the same response. However, the proposed technique forces the use of an FPGA and requires a cumbersome pre-deployment *matching* process. Also, it uses asymmetric (public-key) cryptography, which is significantly more expensive than symmetric key cryptography in terms of energy consumption. Ref. [30] utilizes PUF-generated physically-obfuscated (or secret) keys stored in two IC cards, belonging to the doctor (ED) and patient (IMD) respectively, for authentication and key-exchange between the two entities. However, the protocol requires the services of HS during every authentication session, which has several disadvantages such as high overhead at the IMD and HS as well as service disruption in the event of HS' unavailability. Additionally, both the aforementioned works need special hardware, *viz.* an FPGA and an ASIC, and are not amenable to implementation on COTS devices. Ref. [31] presents a survey of several generic authentication and key-exchange protocols. These protocols are designed for a two-entity ecosystem, where the entities can communicate directly with and authenticate each other. On the other hand, as shown in Fig. 2.4, the IMD ecosystem in the current work comprises of three entities, two of which (IMD and HS) do not have any direct connectivity with each other. As a result, these protocols are not directly applicable to the current ecosystem. Another generic PUF-based protocol was described in Ref. [32], which performs authentication and key-exchange between a resource-constrained prover (containing a PUF) and a resource-rich verifier without requiring the PUF's (prover's) challenge-response pairs to be stored at the verifier end. Moreover, it assumes that the verifier has direct connectivity with the trusted-entity. On the contrary, the current ecosystem is based on IMD (verifier) being resource-constrained while ED (prover) being resource-rich.

IMD (verifier) also does not have direct connectivity with HS (trusted-entity), and hence, the protocol described in Ref. [32] is not applicable to the current ecosystem.

5.2.1 Challenges with the Trusted-entity and PUF-based Approaches

As shown in Table 5.1, IMD can only communicate with another entity over a short-range wireless link (*e.g.,* Bluetooth) and may not have direct connectivity with HS.

i) Without direct connectivity with HS, how can IMD authenticate or verify the identity of (untrusted) ED?

Involving the HS for every single authentication session can incur significant energy overheads at the IMD, besides increasing the load on the HS and the network.

ii) How can the overhead associated with the authentication process be reduced at IMD and HS?

iii) If HS is unavailable or not reachable over the network, how can (untrusted) ED be authenticated?

The second requirement of the communication process between IMD and (authentic) ED is associated with maintaining the confidentiality and integrity of the communication data. Specifically, an attacker may be able to eavesdrop on this communication but should not be able to modify or interpret the content of the communication in any manner. Securing the communication channel through encryption is the most common approach to achieve this. However, due to a limited energy budget and resource constraints, IMD is only capable of utilizing symmetric cryptography. Thus, a symmetric key needs to be shared between IMD and ED before encrypted communication between the two entities can begin. This, in turn, gives rise to another challenge.

iv) How to share a symmetric key between IMD and ED over an insecure (open) communication channel?

We now present the design of an end-to-end authentication and key-exchange protocol for the IMD ecosystem to overcome each of the aforementioned challenges while also addressing the various drawbacks of previous trusted-entity and PUF-based approaches.

5.3 An End-to-End Authentication and Key-exchange Protocol for IMDs

Before presenting the details of the protocol, we highlight an important aspect of the communication channel between the entities when they are deployed in the field. Since both ED and HS have high computational and storage resources (Table 5.1), they communicate over a secure channel at all times by utilizing asymmetric (public key) and symmetric cryptography to encrypt the communication. On the other hand, the resource-constrained nature of IMD (Table 5.1) restricts it to the utilization of symmetric cryptography only. As a result, the (initial) communication between IMD and ED is unencrypted, *i.e.,* over an insecure or open channel, until a symmetric key is exchanged between the two entities. Note that it is this key exchange over an insecure channel that forms one of the key challenges in the design of the proposed protocol.

We now present a detailed description of the protocol, which is divided into two phases – (i) *enrollment phase* and (ii) *authentication phase*. Table 5.2 summarizes the notations and terminology used in the protocol. As shown in Fig. 5.1, IMD and ED are each identifiable by a tuple – $\{public\ id,\ secret\ id\}$. While *public id* (*e.g.,* manufacturer serial number) may be known to other entities, *secret id* is known only to the entity itself (and HS, explained later), thereby helping to prevent *impersonation attacks* (Section 5.4.3). Traditional methods of implementation for the *secret id* would involve storing a unique number in tamper-proof storage such as secure non-volatile memories (NVMs). However, in the current work, *secret id* generation is enabled by a PUF (Section 5.3.3), which generates it on-demand (when needed) instead of statically storing it. This provides better security without the need for additional

Table 5.2.: Notations and terminology used in the protocol

Notation	**Description**	**Length**
P_{IMD}, S_{IMD}	*Public Id*, *Secret Id* of IMD	128 bit
P_{ED}, S_{ED}	*Public Id*, *Secret Id* of ED	128 bit
TS	*Timestamp*	256 bit
L_{IMD}	*IMD salt* stored in HS Database	128 bit
iL_{IMD}	*IMD salt* stored in IMD Database	128 bit
oL_{IMD}	*Old IMD salt*	128 bit
nL_{IMD}	*New IMD salt*	128 bit
L_{ED}	*ED salt*	128 bit
hS_{ED}	*ED verification token*	256 bit
R_{IMD}	Random Number generated by IMD	256 bit
hR_{IMD}	*IMD random token*	256 bit
hR_{ED}	*ED random token*	256 bit
$Auth_{ED}$	*ED's Authentication Outcome*	1 bit
ACK	*Acknowledgement Message*	256 bit
M	Encrypted Message sent by HS	-
D	Decrypted Message at IMD	-
K	Symmetric Key exchanged between IMD and ED	128 bit
Hash(A,B,...)	SHA256 (A xor B xor ...)	256 bit
GenerateSalt()	Generation of Random Number	128 bit
Enc(key, [msg])	Symmetric Encryption of 'msg' with 'key' using AES-128	-
Dec(key, [msg])	Symmetric Decryption of 'msg' with 'key' using AES-128	-

resources (secure NVMs) and also makes the overall design better suited for COTS systems. At the same time, note that the protocol design is agnostic to the *secret id* implementation and can work equally well with either of the implementations mentioned above. Also, as shown in Fig. 5.1, each of the entities – IMD, ED, and HS, also maintain a (private) database at their ends. These databases contain several fields

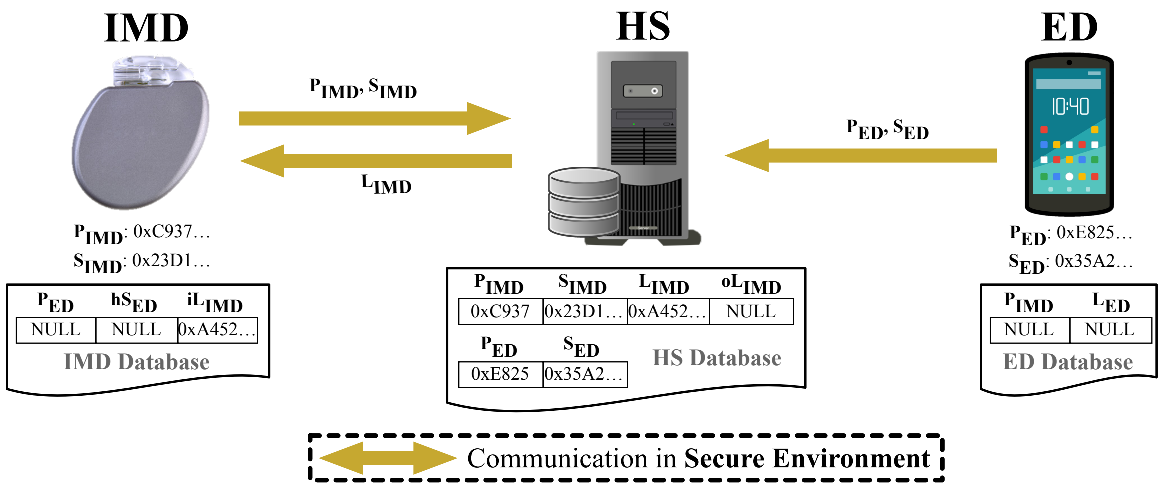


Fig. 5.1.: Enrollment Phase

that are used throughout the protocol and are described in detail in the following paragraphs.

5.3.1 Enrollment Phase

During the first phase in the protocol, IMD and ED are both *enrolled* with HS, as shown in Fig. 5.1. It primarily involves IMD and ED sharing their credentials with HS in a secure environment; these credentials are used by HS to verify the identity of IMD or ED during the following *authentication phase*. *IMD enrollment* is performed by the doctor prior to surgical implantation in the patient. As shown, IMD shares its *public id* (P_{IMD}) and *secret id* (S_{IMD}) with HS, which stores it in its database. Similarly, *ED enrollment* is performed by the doctor or patient before ED begins to operate in the IMD ecosystem. It involves ED sharing its *public id* (P_{ED}) and *secret id* (S_{ED}) with HS, which stores it in its database.

Apart from credentials sharing, certain initialization steps are also performed during the *enrollment phase*. During *IMD enrollment*, HS generates and shares a random number, referred to as *IMD salt*, with IMD, which stores it in its database. Note that *IMD salt* is stored in the L_{IMD} and iL_{IMD} fields at HS and IMD, respectively. Also, as shown in Fig. 5.1, a few other fields in the entities' databases are set to $NULL$ during *IMD/ED enrollment*. For better understanding, we defer the explanation of all these fields to the following section. Thus, at the end of the *enrollment phase*, the databases at the three entities contain some shared credentials and initialized fields, which are shown through an example in Fig. 5.1. Next, we present the second phase of the protocol, *i.e.,* the *authentication phase*.

5.3.2 Authentication Phase

Any new or untrusted device (ED) trying to communicate with IMD must first be authenticated. As mentioned earlier, the proposed protocol follows the trusted-entity approach and utilizes HS to verify the identity of and authenticate ED. We refer to this process as *authentication by HS*. On a day-to-day basis, however, we envision that IMD will only communicate with a few EDs (*e.g.,* doctor's programmer or patient's smartphone) that have been previously authenticated by HS. Though another authentication session may seem redundant for such previously-authenticated devices, it is still necessary to prevent certain sophisticated attacks (*e.g.,* AES attacks, Section 5.4.3). However, invoking the services of HS, which may be catering to several thousand IMDs, for every authentication session (as in [30]) may lead to increased load on HS, thereby affecting the quality of the authentication service. Most importantly, it may lead to increased authentication overhead at IMD, which operates on a very low energy budget (Table 5.1). Hence, we propose that the subsequent authentications be performed by IMD itself, instead of HS. This process, referred to as *authentication by IMD*, builds on top of *authentication by HS* and is an integral part of the protocol design. As shown later, it enables authentication of (untrusted) ED without HS' involvement and helps in substantially reducing the authentication overhead at IMD (Section 5.4). Also, note that the symmetric-key exchange between IMD and ED to encrypt all further communication between the two entities also occurs during *authentication by IMD*. We now describe each of these processes in detail.

(i) Authentication by HS: As mentioned before, IMD can only communicate with another entity over a short-range wireless link (*e.g.,* Bluetooth) and, hence, may not have direct connectivity with HS. This forms the basis of the first challenge described earlier in Section 5.2.1. In the current design, we overcome this challenge in two parts. First, we leverage ED's ability to communicate with both IMD and HS and use ED as a *hop point* to simply forward the messages when IMD and HS send messages to each other. Second, we design the protocol around this leverage and employ measures to guarantee the integrity and confidentiality of these messages, which could be altered by ED. We now describe *authentication by HS* in detail.

Fig. 5.2 shows a concise step-by-step flowchart of *authentication by HS*; each of the steps is further expanded in Fig. 5.3.

- **1-2**: As shown, a new or unauthenticated device (ED) begins the session by sending an access request and its *public id* ($\mathrm{P_{ED}}$) to IMD.
- **3**: IMD checks for ED's *public id* in its database. A valid entry means that ED has been authenticated previously and the session shifts to *authentication by IMD* (described in the next section). Otherwise, it continues as per the following steps.
- **4-5**: IMD generates *timestamp* (TS) and computes its hash with *IMD salt* ($\mathrm{iL_{IMD}}$) and its *secret id* ($\mathrm{S_{IMD}}$) to generate *IMD random token* ($\mathrm{hR_{IMD}}$). Note that a random number could be used in place of *timestamp* if a real-time clock is unavailable at IMD.
- **6-8**: Both *timestamp* and *IMD random token* are sent to ED, which forwards this information to HS alongside IMD's *public id* ($\mathrm{P_{IMD}}$), its own *public id* ($\mathrm{P_{ED}}$), and its own *secret id* ($\mathrm{S_{ED}}$).
- **9-10**: HS first ensures the *freshness* of the message by checking the validity of *timestamp.* It then checks for ED's credentials ($\mathrm{P_{ED}}$, $\mathrm{S_{ED}}$) and IMD's credential ($\mathrm{P_{IMD}}$) in its database. An invalid entry against any of these credentials implies

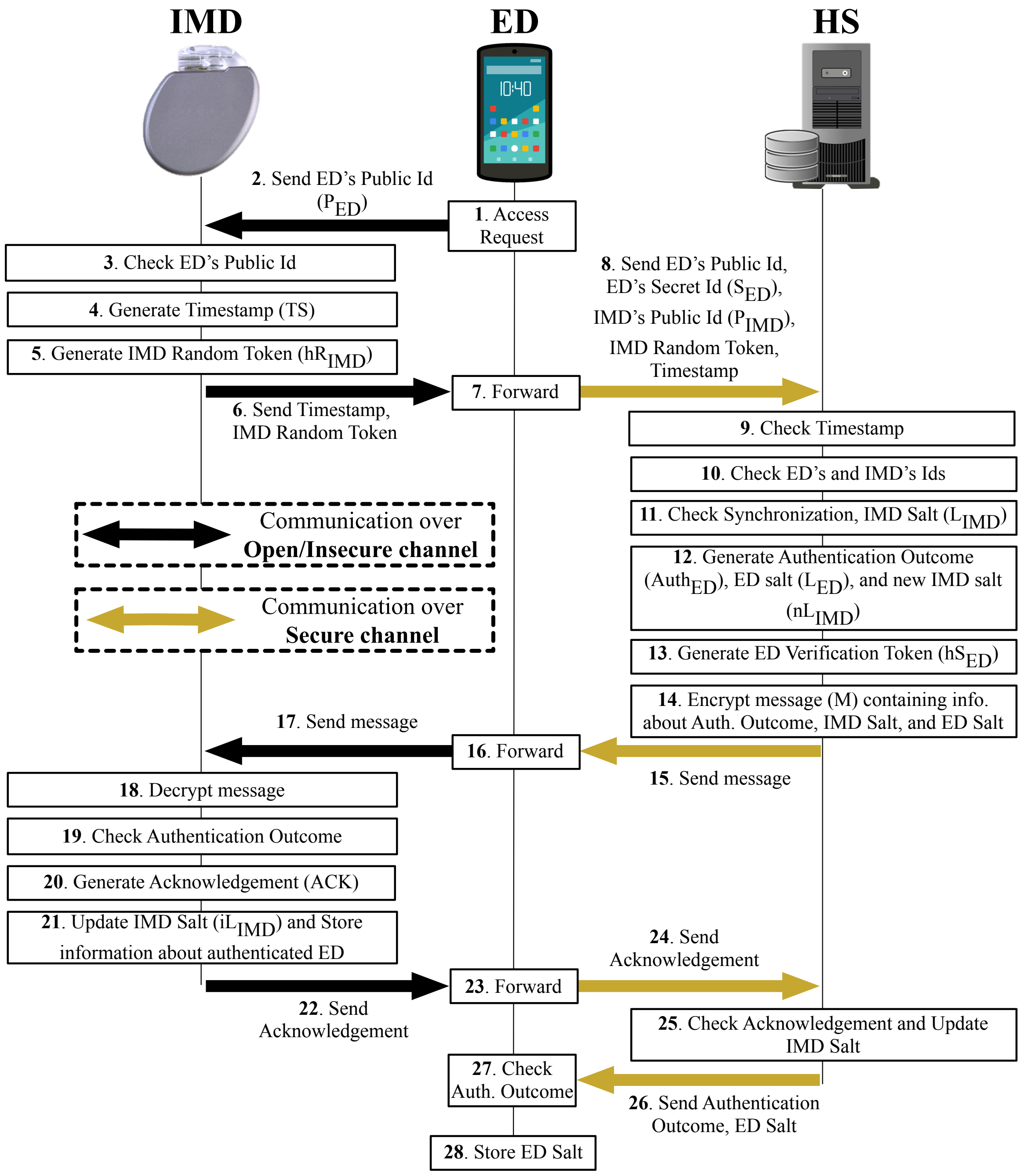

IMD
ED
HS
1. Access Request
2. Send ED's Public Id (P_{ED})
3. Check ED's Public Id
4. Generate Timestamp (TS)
5. Generate IMD Random Token (hR_{IMD})
6. Send Timestamp, IMD Random Token
7. Forward
8. Send ED's Public Id, ED's Secret Id (S_{ED}), IMD's Public Id (P_{IMD}), IMD Random Token, Timestamp
9. Check Timestamp
10. Check ED's and IMD's Ids
11. Check Synchronization, IMD Salt (L_{IMD})
12. Generate Authentication Outcome ($Auth_{ED}$), ED salt (L_{ED}), and new IMD salt (nL_{IMD})
13. Generate ED Verification Token (hS_{ED})
14. Encrypt message (M) containing info. about Auth. Outcome, IMD Salt, and ED Salt
15. Send message
16. Forward
17. Send message
18. Decrypt message
19. Check Authentication Outcome
20. Generate Acknowledgement (ACK)
21. Update IMD Salt (iL_{IMD}) and Store information about authenticated ED
22. Send Acknowledgement
23. Forward
24. Send Acknowledgement
25. Check Acknowledgement and Update IMD Salt
26. Send Authentication Outcome, ED Salt
27. Check Auth. Outcome
28. Store ED Salt
Communication over Open/Insecure channel
Communication over Secure channel

Fig. 5.2.: Authentication by HS

that at least one of the entities (IMD, ED) is not *enrolled* (Section 5.3.1), leading to *authentication failure* and termination of the session.

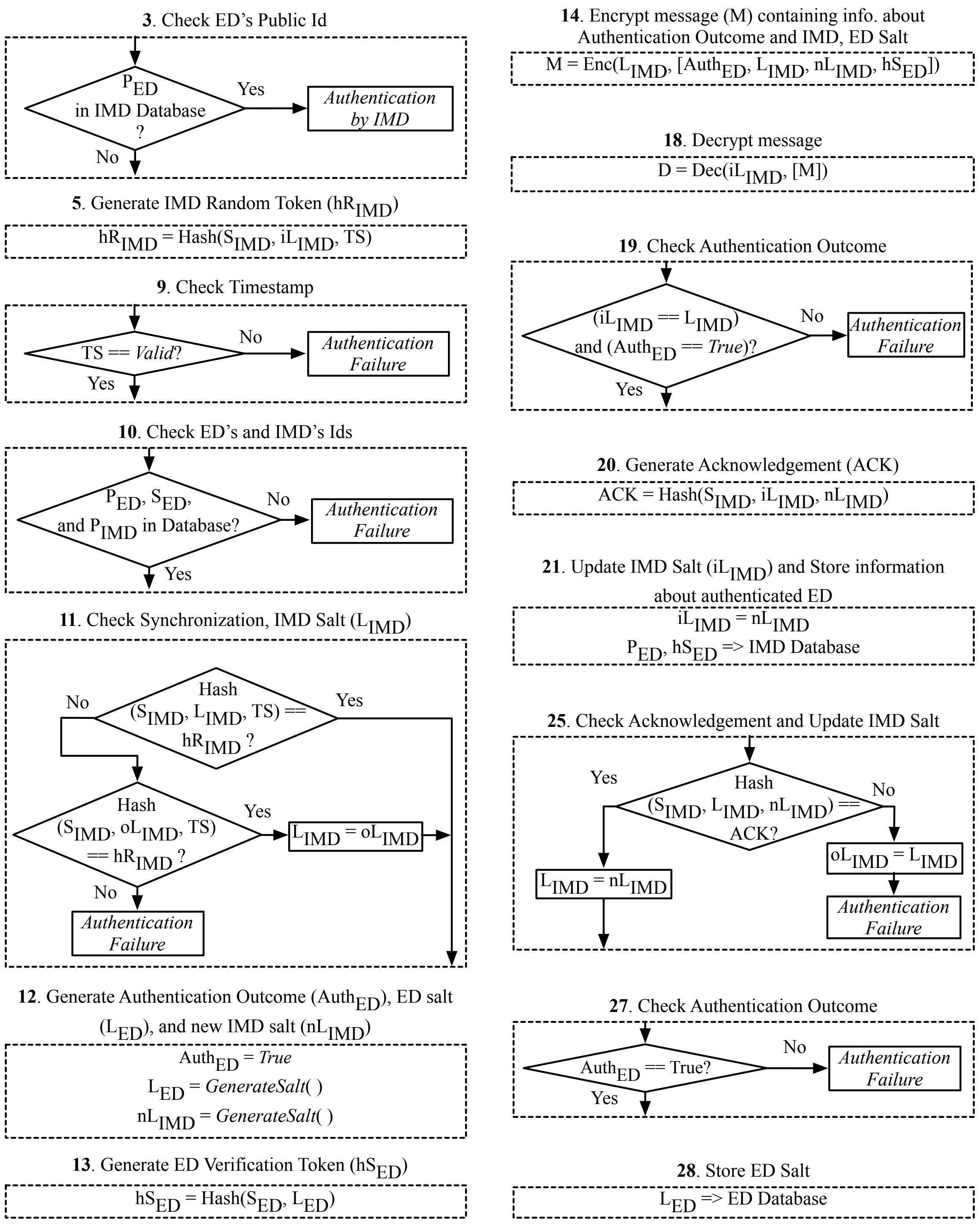

3. Check ED's Public Id
P_ED in IMD Database ?
Yes
No
Authentication by IMD
5. Generate IMD Random Token (hR_IMD)
hR_IMD = Hash(S_IMD, iL_IMD, TS)
9. Check Timestamp
TS == Valid?
No
Yes
Authentication Failure
10. Check ED's and IMD's Ids
P_ED, S_ED, and P_IMD in Database?
No
Yes
Authentication Failure
11. Check Synchronization, IMD Salt (L_IMD)
Hash (S_IMD, L_IMD, TS) == hR_IMD ?
No
Yes
Hash (S_IMD, oL_IMD, TS) == hR_IMD ?
Yes
L_IMD = oL_IMD
No
Authentication Failure
12. Generate Authentication Outcome (Auth_ED), ED salt (L_ED), and new IMD salt (nL_IMD)
Auth_ED = True
L_ED = GenerateSalt( )
nL_IMD = GenerateSalt( )
13. Generate ED Verification Token (hS_ED)
hS_ED = Hash(S_ED, L_ED)
14. Encrypt message (M) containing info. about Authentication Outcome and IMD, ED Salt
M = Enc(L_IMD, [Auth_ED, L_IMD, nL_IMD, hS_ED])
18. Decrypt message
D = Dec(iL_IMD, [M])
19. Check Authentication Outcome
(iL_IMD == L_IMD) and (Auth_ED == True)?
No
Yes
Authentication Failure
20. Generate Acknowledgement (ACK)
ACK = Hash(S_IMD, iL_IMD, nL_IMD)
21. Update IMD Salt (iL_IMD) and Store information about authenticated ED
iL_IMD = nL_IMD
P_ED, hS_ED => IMD Database
25. Check Acknowledgement and Update IMD Salt
Hash (S_IMD, L_IMD, nL_IMD) == ACK?
Yes
No
L_IMD = nL_IMD
oL_IMD = L_IMD
Authentication Failure
27. Check Authentication Outcome
Auth_ED == True?
No
Yes
Authentication Failure
28. Store ED Salt
L_ED => ED Database

Fig. 5.3.: Steps in Authentication by HS

- **11**: HS reads IMD's *secret id* ($\mathrm{S_{IMD}}$) as well as *IMD salt* ($\mathrm{L_{IMD}}$) and *old IMD salt* ($\mathrm{oL_{IMD}}$) fields from its database. At this stage, we take a short detour and highlight the role of the last two fields. If *IMD salt* stored at IMD database ($\mathrm{iL_{IMD}}$) and HS database ($\mathrm{L_{IMD}}$) have the same value, then the two entities are said to be in a *synchronized state*. As will be evident shortly, this synchronization between the two entities enables them to verify the integrity as well as maintain the confidentiality of the messages exchanged between them using the (common) *IMD salt*; note that these messages are forwarded by (untrusted) ED. However, over the course of several sessions, IMD and HS may reach a *desynchronized state*, *i.e.,* they may have different *IMD salt* values in their database. In such a scenario, *old IMD salt*, which holds the *IMD salt* value corresponding to the last *synchronized state*, is utilized to synchronize the two entities again. Hence, after reading these fields from its database, HS computes a hash of IMD's *secret id* and *timestamp* with the *IMD salt* and compares it with (the received) *IMD random token* ($\mathrm{hR_{IMD}}$). A successful match verifies the integrity of the message sent by IMD as well as synchronization between the two entities. An unsuccessful match prompts HS to compute the above hash with *old IMD salt* (instead of *IMD salt* earlier), followed by a comparison with *IMD random token*. A successful match implies that the two entities are in a *desynchronized state* and the value of *IMD salt* in HS database is updated with that of *old IMD salt* in order to synchronize them again. An unsuccessful match, on the other hand, implies that the integrity of the message is compromised, resulting in *authentication failure* and termination of the session.

- **12-13**: HS sets *ED's authentication outcome* ($\mathrm{Auth_{ED}}$) to *true*. Next, it generates two random numbers – *new IMD salt* ($\mathrm{nL_{IMD}}$) and *ED salt* ($\mathrm{L_{ED}}$). It also generates *ED verification token* ($\mathrm{hS_{ED}}$) by computing the hash of ED's *secret id* with *ED salt*. While *new IMD salt* is used (later, if the authentication is successful) to update *IMD salt*, *ED salt* and *ED verification token* are used during *authentication by IMD* and are explained in the next section.

- **14-17**: HS constructs a *plaintext* composed of *ED's authentication outcome*, *IMD salt*, *new IMD salt*, and *ED verification token*. Next, in order to ensure the confidentiality of the message, it encrypts the *plaintext* using *IMD salt* as the symmetric key. This encrypted message (M) is then sent to ED, which forwards it to IMD. Note that due to the encrypted nature of the message, ED has no visibility into its contents.

- **18**: Upon receiving the encrypted message, IMD decrypts it using *IMD salt* ($\mathrm{iL}_{\mathrm{IMD}}$, stored in its database) as the symmetric key.

- **19**: IMD first checks the integrity of the message by comparing *IMD salt* in the decrypted message with *IMD salt* in its database. Next, it checks if *ED's authentication outcome* is *true*. Not satisfying any of these conditions leads to *authentication failure* and termination of the session.

- **20**: IMD generates *acknowledgement message* (ACK) by hashing its *secret id* with *IMD salt* and *new IMD salt*.

- **21-24**: IMD updates the value of *IMD salt* with that of *new IMD salt*. Next, it stores the ED's *public id* and *ED verification token* in its database for use during *authentication by IMD*. Then, it sends *acknowledgement message* to ED, which forwards it to HS.

- **25**: HS generates a hash value of IMD's *secret id*, *IMD salt*, and *new IMD salt*, and compares it with received *acknowledgement message*. Note that this step also checks for synchronization between IMD and HS; an absence of or corrupted *acknowledgement message* implies that there could be possible desynchronization between the two entities. In such a scenario, it cannot be inferred with certainty whether *IMD salt* was successfully updated at IMD's end. Hence, HS stores the *IMD salt* value in *old IMD salt* and causes *authentication failure* followed by session termination. A verified acknowledgment, on the other hand, causes HS to update the value of *IMD salt* with that of *new IMD salt*.

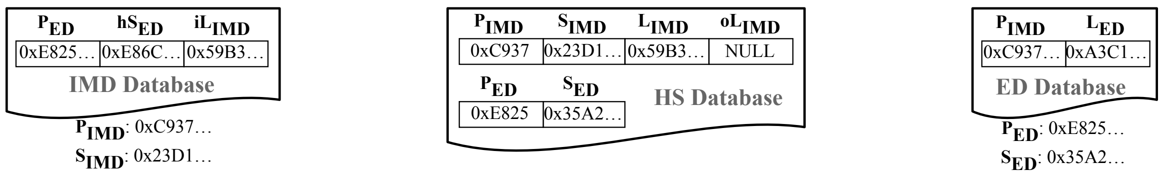


Fig. 5.4.: Database after Authentication by HS

- **26-28**: HS sends *ED authentication outcome* and *ED salt* to ED. At its end, ED first checks for *ED authentication outcome* to confirm successful authentication and then stores *ED salt* alongside IMD's *public id* in its database for use during *authentication by IMD*.

Fig. 5.4 shows an example picture of the databases at the three entities at the end of a successful *authentication by HS* session. At this stage, the first challenge, which was described earlier in Section 5.2 and is associated with ED authentication without direct connectivity between IMD and HS, has been addressed. In the next paragraph, we present *authentication by IMD* and discuss how it addresses the next three challenges, which are associated with – reduction of the overhead in the authentication process at IMD (and HS), authentication of ED without HS' involvement, and key exchange between IMD and ED over an insecure channel.

(ii) Authentication by IMD: On a day-to-day basis, we envision that IMD will only communicate with a few EDs (*e.g.,* doctor's programmer or patient's smartphone) that have been previously authenticated by HS. Though another authentication session may seem redundant for such previously-authenticated EDs, it is still necessary to prevent certain sophisticated attacks (Section 5.4.3). As mentioned earlier, these subsequent authentications are performed by IMD itself through the process referred to as *authentication by IMD*. This process also includes the symmetric-key exchange between IMD and ED that allows encryption of all further communication between the two entities. Hence, *authentication by IMD* is performed under two scenarios. First, it follows directly after *authentication by HS*, *i.e.,* an unknown ED, which has just undergone *authentication by HS*, is requesting the exchange of the first symmetric key. Second, a known ED, which has undergone *authentication by IMD* in the past, is requesting access to IMD directly. The second scenario is particularly important and it forces even a known ED to undergo authentication and then exchange a new key with IMD; this helps to prevent certain sophisticated attacks (*e.g.,* AES attacks), as described later in Section 5.4.3. At the same time, by enabling IMD to perform the authentication and key exchange (instead of HS), the *authentication by IMD* process reduces the load on HS as well as the authentication overhead at IMD (Section 5.4). Fig. 5.5 shows a concise step-by-step flowchart of *authentication by IMD*; each of the steps is further expanded in Fig. 5.6. Note that the process is based around IMD and ED only and leaves out HS. We now describe the steps in detail.

- **1-2**: As shown, ED begins the session by sending an access request and its *public id* (P_{ED}) to IMD.
- **3-5**: IMD checks for ED's *public id* in its database. An invalid entry implies that ED has not been authenticated previously and the session shifts to *authentication by HS* (described earlier). A valid entry, on the other hand, prompts IMD to generate a random number (R_{IMD}) and send it to ED.
- **6-7**: Upon receiving the random number, ED first computes *ED verification token* (hS_{ED}) by hashing its *secret id* (S_{ED}) with *ED salt* (L_{ED}). *ED verification token*, in turn, is hashed with the random number to generate *ED random token* (hR_{ED}), which is then sent to IMD.
- **8-10**: At its end, IMD performs a hash of the generated random number with *ED verification token*, which is stored in its database, to compute *IMD random token* (hR_{IMD}). *IMD random token* is then compared with (received) *ED random token.* A successful match verifies the identity of ED and prompts IMD to set *ED authentication outcome* ($Auth_{ED}$) to *true.* An unsuccessful match, on the contrary, causes *authentication failure* and termination of the session. Finally, IMD sends *ED authentication outcome* to ED.

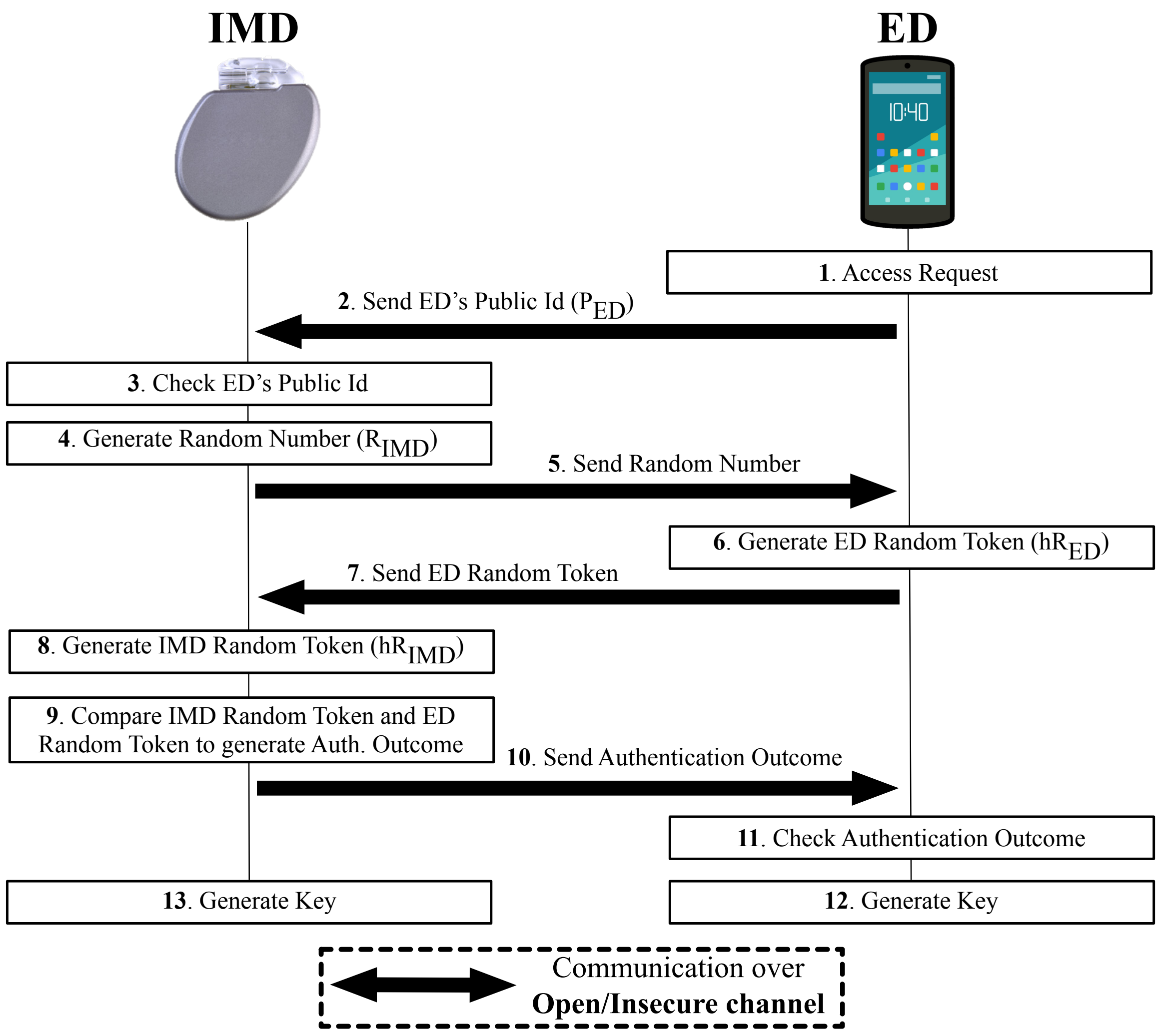

IMD
ED
10:40
1. Access Request
2. Send ED's Public Id (P_{ED})
3. Check ED's Public Id
4. Generate Random Number (R_{IMD})
5. Send Random Number
6. Generate ED Random Token (hR_{ED})
7. Send ED Random Token
8. Generate IMD Random Token (hR_{IMD})
9. Compare IMD Random Token and ED Random Token to generate Auth. Outcome
10. Send Authentication Outcome
11. Check Authentication Outcome
13. Generate Key
12. Generate Key
Communication over
Open/Insecure channel

Fig. 5.5.: Authentication by IMD

At this stage, we want to highlight the roles of *ED verification token* (hS_{ED}) and *ED salt* (L_{ED}). As mentioned earlier, during *authentication by HS*, *ED verification token* is stored in IMD database. Subsequently, during *authentication by IMD*, ED uses it to compute *ED random token*, which is compared against *IMD random token* (computed by IMD) to ultimately decide whether ED is authenticated or not. Thus, *ED verification token* essentially behaves as the basis upon which IMD verifies the authenticity of ED during *authentication by IMD*. Here, one could propose an alternative method *i.e.,* store ED's *secret*

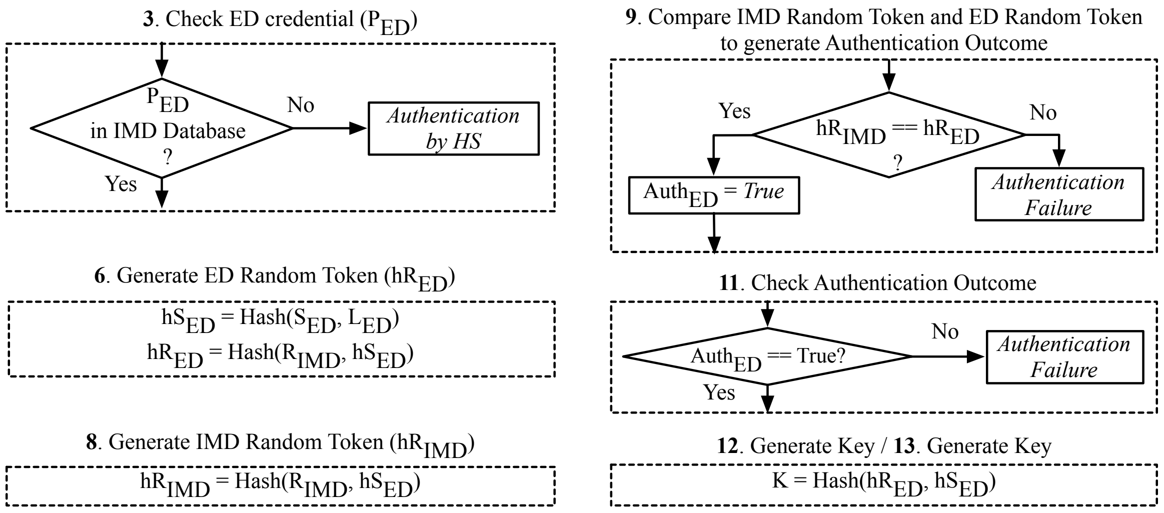

3. Check ED credential (P_ED)
P_ED in IMD Database ?
No
Authentication by HS
Yes
6. Generate ED Random Token (hR_ED)
hS_ED = Hash(S_ED, L_ED)
hR_ED = Hash(R_IMD, hS_ED)
8. Generate IMD Random Token (hR_IMD)
hR_IMD = Hash(R_IMD, hS_ED)
9. Compare IMD Random Token and ED Random Token to generate Authentication Outcome
Yes
hR_IMD == hR_ED ?
No
Auth_ED = True
Authentication Failure
11. Check Authentication Outcome
Auth_ED == True?
No
Yes
Authentication Failure
12. Generate Key / 13. Generate Key
K = Hash(hR_ED, hS_ED)

Fig. 5.6.: Steps in Authentication by IMD

id in IMD's database (instead of *ED verification token*) and utilize it the same way as *ED verification token.* While the functionality of the protocol is not affected by this alternate method, it may pose a security concern resulting from the storage of ED's *secret id* in raw form in IMD database. Specifically, if the *secret id* gets leaked or is retrieved from the database, it could potentially lead to impersonation attacks on the other IMDs that also communicate with the (same) ED. Hence, ED's *secret id* is always stored in a hashed form in IMD's database; *ED salt* is utilized here to compute this hash, as described earlier.

- **11-13 (Key exchange)**: ED first checks the authentication outcome to confirm successful authentication. Next, both IMD and ED compute the key, K, at their ends by performing a hash of *ED random token* and *ED verification token.* This key-exchange step concludes *authentication by IMD* and all further communication between the two entities is encrypted using K.

In the next section, we present a brief discussion about the generation of *secret ids* and random numbers utilized in the proposed protocol.

5.3.3 Generation of Secret IDs and True Random Numbers

From the previous sections, it is evident that *secret ids* and random numbers play an important role in the proposed protocol. As described earlier, traditional methods of implementation for *secret ids* would involve storing unique numbers in tamper-proof storage such as secure non-volatile memories (NVMs). Similarly, the generation of (true) random numbers would require dedicated/custom hardware on the chip. Hence, in the current work, the generation of both *secret ids* and random numbers is enabled by an (on-chip) PUF, which generates them on demand. This provides better security without the need for additional resources (secure NVMs, *etc.*) and makes the overall design better suited for COTS systems. At the same time, note that the core functionality of the proposed protocol is agnostic to the *secret id* and random number implementation and can work equally well with either of the implementations mentioned above.

Next, we present the details related to the implementation and analysis of the proposed protocol.

5.4 Implementation and Analysis

In this section, we first present the implementation details of the prototype IMD ecosystem and the experimental setup used to evaluate the proposed protocol. Next, we perform an exhaustive evaluation of the overheads at IMD during authentication followed by a security analysis of the protocol.

5.4.1 Prototypes and Experimental Setup

Fig. 5.7 presents the prototype implementations of IMD, ED, and HS used in the proposed protocol as well as the experimental setup used. The IMD is implemented using a Silicon Labs TG11 Starter Kit [78] containing an ARM Cortex-M0+ based microcontroller (MCU) with 128KB Flash and 32KB RAM and is clocked at 48MHz.

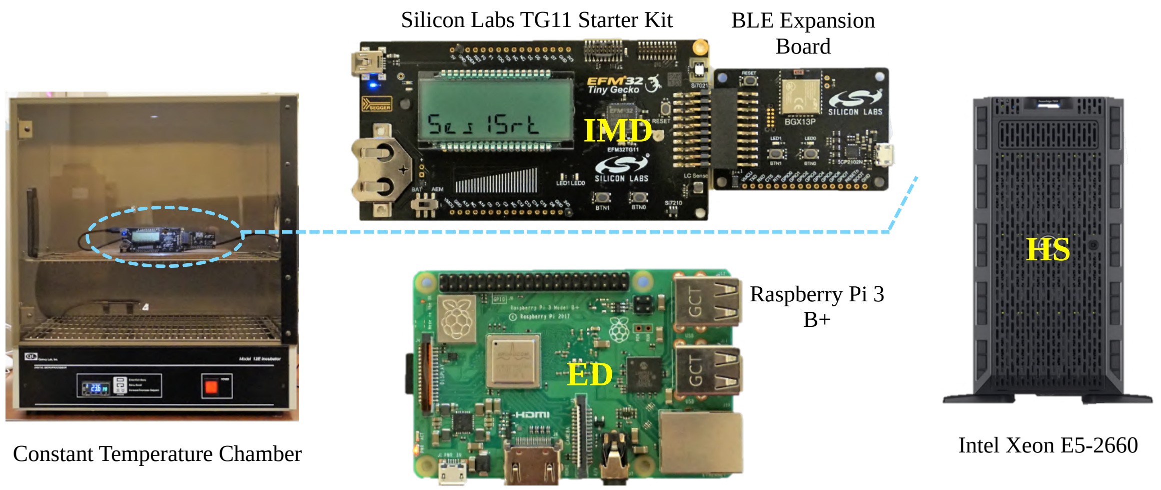


Fig. 5.7.: Prototype IMD Ecosystem and Experimental Setup

We use a Silicon Labs BGX13P Bluetooth Low Energy (BLE) Expansion Board [79] to provide Bluetooth connectivity to the IMD, which communicates with the MCU board via a serial interface at a baud-rate of 9600 bps. The ED is implemented on a Raspberry Pi 3 Model B+ board [80], which contains a Cortex-A53 based SoC and communicates with the IMD over BLE. Finally, the HS is implemented on a server based on the Intel Xeon E5-2660 processor. The ED and the HS communicate over Wi-Fi and Ethernet in the current implementation. The energy consumption and latency associated with the protocol are measured using the Silicon Labs Multi-Node Energy Profiler [81], which utilizes the Advanced Energy Monitor on-board the TG11 starter kit to take real-time current and latency measurements. Experiments related to varying temperatures were performed by placing the IMD inside a constant temperature chamber.

Although other types of PUFs [55] could be utilized in the proposed protocol, we chose a memory-based PUF [8, 9, 40, 41, 48, 51, 82–84] to implement in our prototype design. Specifically, we utilize an SRAM PUF [85] that leverages the *power-cycling* (power off $\rightarrow$ power on $\rightarrow$ read SRAM) approach to generate unique *startup values*. These values, generated from 32-bit words, are hashed (*e.g.,* using SHA-256) to produce *secret ids* and true random numbers on demand. The choice of utilizing the *power-cycling* based SRAM PUF stems from the following. First, SRAM is ubiquitous in most embedded systems, and hence, the PUF requires minimal or no additional resources for its implementation and operation. Second, a *power-cycling* based SRAM PUF exhibits high entropy making it suitable for the generation of *secret ids* and random numbers [85]. A lightweight algorithm is utilized to systematically identify locations in the SRAM that can serve as sources of *secret ids* and random numbers; the algorithm is described later in Section 5.5.1. The SRAM PUF in the IMD is implemented on an unused 8 KB SRAM block in the TG11 MCU through the *power-cycling* approach. On the Raspberry Pi (ED), access to the SRAM (for reading startup value) is restricted by the operating system. Hence, for the prototype implementation, the PUF operation was emulated by reading pre-stored 128-bit strings from its flash memory.

Finally, for performing the hash (SHA-256) and encryption/decryption (AES-128) operations in the IMD, we explored two possible implementations – HW-based and SW-based, which utilize a hardware cryptography engine on the MCU and regular MCU instructions, respectively.

5.4.2 Overhead Analysis at the IMD

The proposed protocol is designed to incur minimal overhead at the IMD, as shown in Table 5.3. The total overhead is classified into three types – *energy*, *latency*, and *memory*. For *energy* and *latency*, there is a negligible difference in the HW-based and SW-based implementations, and hence, Table 5.3 lists a single value for these overheads for both the implementations. This is because an entire authentication session in the proposed protocol only contains a small number of these cryptographic operations, as a result of which, their net contributions to the overall overheads are very small.

Next, *energy* and *latency* are each divided into three parts – *computation*, *communication*, and *sleep*, based on the source of the overhead. While the first two sources

Table 5.3.: Protocol Overhead at the IMD

Overhead		**Authentication by IMD**	**Authentication by HS**
Energy (μJ)	Computation	52	81
	Communication	4194	12589
	Sleep	22	23
	Total	**4268**	**12693**
Latency (ms)	Computation	8	11
	Communication	177	528
	Sleep	1010	1040
	Total	**1195**	**1579**
Memory (bytes)	RAM (data)	248 (*HW*), 504 (*SW*)	
	Flash (text)	33K (*HW*), 37K (*SW*)	

are self-explanatory, *sleep* refers to the MCU's low-energy state when it is waiting to receive data from the ED or HS. As shown, the largest shares of the *energy* and *latency* overheads are attributed to *communication* and *sleep*, respectively, while *computation* makes a relatively much smaller contribution. Also, the overheads (specifically, *latency*) due to *sleep* could vary depending upon the latency of the communication with the other entities (HS and ED) as well as the tasks performed at their ends. Overall, the total energy consumed by the protocol is very low – 4268 μJ and 12693 μJ for a single session of *authentication by IMD* and *authentication by HS*, respectively. To put this into perspective, we consider a typical IMD [20] that has a lifetime of 90 months and is powered by a 2 Ah battery; the latter translates to an energy budget of 23760 J (at 3.3V supply). Therefore, a single session of *authentication by IMD* and *authentication by HS* adds an overhead of only 0.000018% and 0.000053% of the IMD's total energy budget, respectively. Specifically, even with ten authentication sessions per day, the total energy overhead incurred by the protocol over the lifetime of the IMD is 115 J and 342 J, *i.e.,* only 0.5% and 1.4% of its total energy budget, for *authentication by IMD* and *authentication by HS*, respectively. Moreover, *authentication by IMD* is both more energy-efficient (by 66%) and faster (by 24%) as compared to *authentication by HS*, thus satisfying an important design requirement of the protocol. We also present the memory footprint of the program executing on the IMD. As shown, (248 bytes RAM, 33 KB Flash) and (504 bytes RAM, 37 KB Flash) are used by the `.data` and `.text` sections of the program in the HW-based and SW-based implementations, respectively.

5.4.3 Security Analysis

In this section, we first present the attacker model, followed by a detailed analysis of the protocol against security and privacy attacks.

Attacker Model: We assume that an attacker has the ability to passively eavesdrop on the communication between the IMD and ED as well as actively jam or corrupt the corresponding packets. However, packets exchanged between the ED and HS are encrypted at all times and are not readable/corruptible by the attacker.

Eavesdropping Attacks: An attacker can passively listen to the unencrypted messages between IMD and ED, revealing only their respective public identities (*public ids*). All other messages exchanged between the two entities are either encrypted, randomly generated, in a hashed form, or contain general information such as the authentication outcome, all of which do not reveal any useful information to the attacker.

Replay Attacks: In order to gain access or reveal relevant information about the entities, an attacker can replay messages such as *IMD random token* (hR_{IMD}), *M*, and ACK during *authentication by HS* and *ED random token* (hR_{ED}) during *authentication by IMD*. The protocol employs *timestamp* at the beginning of the protocol to prevent the replay of *IMD random token*. *M* and ACK are closely tied to the current value of *IMD salt* (L_{IMD}) and/or *new IMD salt* (nL_{IMD}) and, hence, replaying any of these would also cause *authentication failure*. Similarly, *ED random token* is closely tied with *ED salt* (L_{ED}) and the random number – R_{IMD}, hence, replaying it would cause *authentication failure*.

Impersonation Attacks: An attacker can impersonate an authenticated ED or a legitimate IMD by using their respective *public ids* and initiate communication with the other entity. However, further communication is restricted in such events through the use of *secret ids*, which are checked directly and indirectly (though hash computations) at various stages of the protocol to verify the identity of the communicating entities.

Desynchronization Attacks: An attacker may deliberately cause desynchronization between the IMD and HS by jamming or corrupting the messages – M and ACK. As described earlier, the protocol inherently takes this into consideration and uses the *old IMD salt* field in the HS database to synchronize the two entities again, if required.

Invasive Attacks: A well-equipped attacker with physical access to the entities could mount invasive attacks (*e.g.,* physical tampering) in a bid to extract relevant information (*e.g., secret id*) from the entities' database. Even though obtaining physical access to an IMD that is implanted inside a patient's body is unlikely, the protocol provides substantial protection against such attacks. The owner entity's *secret id* is not stored in any local database but generated on demand by a PUF, which is known to be resistant to invasive attacks. Also, an entity's *secret id* is always stored in a hashed form at any other entity's database, revealing no information about it.

Side-Channel Attacks: In the proposed protocol, a side-channel attack [22] on the IMD and (authenticated) ED could potentially leak critical data, *e.g., secret ids* (S_{IMD}, S_{ED}), *salts* (iL_{IMD}, L_{ED}), *symmetric key* (K), *etc.* The protocol inherently provides high resistance to such an attack. First, the protocol utilizes a PUF for generating *secret ids* on demand. As shown in the next section, the PUF is capable of generating multiple *secret ids*, thereby allowing the protocol to vary them across authentication sessions, if required. Second, the protocol ensures that the other critical data (*salts*, *symmetric key*, *etc.*) are not constant but also change across different sessions.

Together, these design features provide high resistance against side-channel attacks. Additionally, the protocol could be easily modified to include existing techniques [22] to further enhance its strength against such attacks.

Battery-Drain and Denial-of-Service Attacks: Although the protocol doesn't explicitly address these, simple enhancements such as causing a *time-out* or utilizing an alternative communication channel [27] could be made to the proposed design to address battery-drain and denial-of-service attacks, respectively.

Modeling and AES Attacks: By design, the protocol provides substantial protection against modeling [86] and AES attacks [21] through the use of a PUF and (one-way) hash functions, which generate time-varying encryption keys and mask the communication between the entities, respectively. A formal analysis of the protocol's vulnerability to these attacks is beyond the scope of this work and is planned as a future project.

5.5 Additional Discussions and Results

This section presents some additional design aspects, results, and discussions associated with the proposed protocol.

5.5.1 SRAM PUF: Generation of Secret IDs and Random Numbers

We present and describe a lightweight algorithm to systematically identify locations in the SRAM that can serve as sources of *secret ids* and random numbers.

Algorithm 4 starts by employing the *power-cycling* approach to generate the *startup value* of an entire SRAM block (B) several times, specified by the number of runs (N_r). Next, the block is divided into non-overlapping words of size S bits each. For each word, its startup value (extracted from the block's startup value) during a particular run is compared with all the other runs. During this step, a counter keeps track of the number of comparisons in which the startup values differ, *i.e.,* in which their Hamming Distance is non-zero. Finally, the counter value is used to calculate

Algorithm 4: Identification of SRAM Words for *Secret Id* and Random Number Generation

```
Input: $B$ = SRAM Block, $S$ = Word Size, $N_r$ = Number of Runs, $P_{diff} = P$
Output: $W_p$ = Words that exhibit $P_{diff}$ equal to $P$
1  $W_p = \phi$
2  $Count_{comp} = (N_r - 1)(N_r - 2) / 2$
3  for $i = 1$ to $N_r$ do
4      $V_i = Gen\_Startup\_Value\ (B)$
5  $W_a = Gen\_All\_NonOverlap\_Words\ (B, S)$
6  foreach $w \in W_a$ do
7      $Count_{diff} = 0$
8      for $i = 1$ to $N_r$ do
9          $vw_i = Get\_Startup\_Value\_Word\ (w, V_i)$
10         for $j = i + 1$ to $N_r$ do
11             $vw_j = Get\_Startup\_Value\_Word\ (w, V_j)$
12             if $vw_i \neq vw_j$ then
13                 $Count_{diff} = Count_{diff} + 1$
14     $P_{diff} = (Count_{diff} / Count_{comp})$ *100
15     if $P_{diff} == P$ then
16         $W_p = W_p \cup w$
```

Percentage Different or P_{diff}, which represents the percentage of times the word generated a different startup value. Note that a low P_{diff} implies that the word produces a stable startup value across several runs, making it ideal for generating a *secret id*. On the other hand, a high P_{diff} implies an unstable startup value and makes the word suitable for random number generation. The algorithm takes the required P_{diff} as input and produces a list of all the words in the SRAM block that meet this criterion, as will be described next.

We followed the methodology described in Algorithm 4 to generate the start-up of an 8 KB SRAM block and then identify 32-bit words that are suitable for the

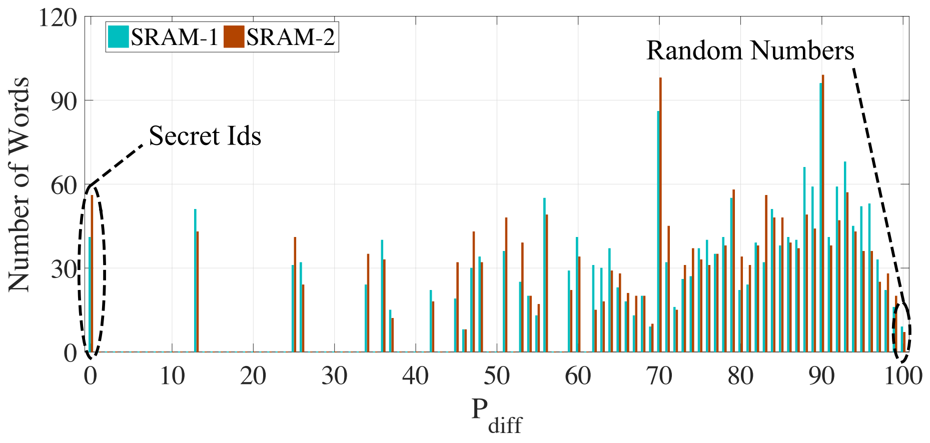


Fig. 5.8.: Secret IDs and Random Numbers in the SRAM PUF

generation of *secret ids* and random numbers. The results of this process are shown in Fig. 5.8, which shows a distribution of the number of words that exhibit a certain (specified) *Percentage Different* (or $\mathrm{P_{diff}}$). Note that, to rule out any bias due to a specific SRAM module, we performed the process on two SRAM modules (on-board two different IMD prototypes). Moreover, an IMD is expected to operate across the entire range of body temperature, which could vary between 34°C – 42°C. Hence, the startup values were generated at three (sample) temperature points – 34°C, 37°C, and 42°C, with (a sample) ten runs at each point. All these startup values were then analyzed (together) to identify words that exhibit the specified $\mathrm{P_{diff}}$. As shown, 41 and 56 words exhibit $P_{\mathrm{diff}} = 0\%$ in SRAM-1 and SRAM-2, respectively. The startup values generated by these words remained constant across every run and temperature point and, thus, can be used as *secret ids*. Similarly, 7 and 9 words exhibit $P_{\mathrm{diff}} = 100\%$ in SRAM-1 and SRAM-2, respectively. In other words, each of these generated a different startup value across every run and temperature point and can be used as a source of true random numbers. Note that slightly relaxing the criterion, *e.g.*, $P_{\mathrm{diff}} = 12\%$ or 90%, could help meet the requirements of more *secret ids* and random numbers, as shown in Fig. 5.8; Algorithm 4 (and the SRAM PUF) enables this but the decision is ultimately left to the end-user.

Note that the number of runs and temperature points used above represent a sample-case only; if required, a more rigorous analysis could be performed through Algorithm 4 using more runs and temperature points. Also, the uniqueness and randomness associated with the startup values of a *power-cycling* based SRAM PUF are well-established [85, 86], hence, we do not present any such analysis in this work.

5.5.2 Emergency Access to the IMD

For any authentication protocol involving IMDs, support for *emergency access* (to the IMD) is an important (for safety) but conflicting (for security) requirement. Traditionally, *master keys* [19], which are generated by the HS and shared with both the IMD (during *enrollment phase*) and emergency personnel, have been used for this. Although this technique could be easily integrated with the proposed protocol to provide emergency access, we are currently exploring two approaches which are potentially more secure than the use of *master keys*. The first approach augments the proposed protocol with mechanisms to utilize biometrics or side-channels for providing emergency access. The second approach explores the suitability of digital certificates (and Public Key Infrastructure (PKI)) in an IMD ecosystem towards achieving the same goal.

5.6 Conclusion

This chapter presents a lightweight end-to-end authentication and key-exchange protocol for an IMD ecosystem. The protocol is based on the trusted-entity approach and successfully addresses the unique challenges of performing authentication and key-exchange in the IMD ecosystem. We also present the application of an *intrinsic* PUF, which integrates seamlessly with the cryptography techniques used in the protocol to result in a secure and lightweight solution protecting against various security and privacy attacks. The design is implemented on a real system comprising of several off-the-shelf devices. Experimental evaluation (using metrics such as energy overheads, operation under varying temperature conditions, and protection against security and privacy attacks) demonstrates the effectiveness of the proposed protocol in providing robust, secure, and low-overhead authentication and key-exchange in IMDs.

6. SUMMARY

The last decade has seen a rapid proliferation of embedded computing devices, fueled in part by the advent of the Internet-of-Things (IoT) era. The increasingly network-connected nature of these devices, coupled with their ability to access potentially sensitive or confidential information, has given rise to a plethora of new security and privacy concerns. An additional challenge is the growing number of counterfeit components in these devices, with serious reliability and financial repercussions.

In this dissertation, we explore device authentication as a means to address these challenges. The first part of the dissertation proposes the design of two memory-based Physically Unclonable Functions (PUFs) that enable low-cost and low-overhead device authentication and secure-key exchange in off-the-shelf embedded systems. These PUFs leverage the memory (DRAM, SRAM) in the system, thus, requiring minimal (or no) additional hardware for operation. Two lightweight authentication and error-correction techniques, which ensure robust operation under a range of environmental and temporal variations, are also proposed. Experimental results obtained from prototype implementations demonstrate the effectiveness of both the designs. The second part of the dissertation explores the application of these techniques in real-world systems through a new end-to-end authentication and key-exchange protocol in the context of an Implantable Medical Device (IMD) ecosystem. The protocol is based on the trusted-entity approach and successfully addresses the unique challenges of performing authentication and key-exchange in the IMD ecosystem. The design is implemented on a real system comprising of several off-the-shelf devices. Experimental evaluation using metrics such as energy and latency overheads and analysis against several security and privacy attacks demonstrates the effectiveness of the proposed protocol in providing robust, secure, and low-overhead authentication and key-exchange in IMDs.

In summary, this dissertation takes a significant step towards enabling low-cost and low-overhead device authentication in embedded systems. We hope that the work presented here will help address several key challenges towards the realization of a secure IoT.

REFERENCES

REFERENCES

[1] IHS Markit, "The Internet of Things: a movement, not a market," 2017. [Online]. Available: https://cdn.ihs.com/www/pdf/IoT_ebook.pdf

[2] A. Greenberg, "Hackers Remotely Kill a Jeep on the Highway—With Me in It," 2015. [Online]. Available: https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/

[3] U.S. Food and Drug Administration, "Cybersecurity Vulnerabilities Affecting Medtronic Implantable Cardiac Devices, Programmers, and Home Monitors: FDA Safety Communication," 2019. [Online]. Available: https://www.fda.gov/medical-devices/safety-communications/2019-safety-communications

[4] D. Criss, "Software vulnerabilities in some medical devices could leave them susceptible to hackers, FDA warns," 2019. [Online]. Available: https://www.cnn.com/2019/10/02/health/fda-medical-devices-hackers-trnd/index.html

[5] M. Pecht and S. Tiku, "Bogus: electronic manufacturing and consumers confront a rising tide of counterfeit electronics," *IEEE Spectrum*, vol. 43, no. 5, pp. 37–46, May 2006.

[6] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, "Silicon Physical Random Functions," in *Conference on Computer and Communications Security (CCS).* New York, NY, USA: ACM, 2002, pp. 148–160. [Online]. Available: http://doi.acm.org/10.1145/586110.586132

[7] S. Sutar, A. Raha, and V. Raghunathan, "D-PUF: An Intrinsically Reconfigurable DRAM PUF for Device Authentication in Embedded Systems," in *International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES).* New York, NY, USA: ACM, 2016, pp. 12:1–12:10. [Online]. Available: http://doi.acm.org/10.1145/2968455.2968519

[8] S. Sutar, A. Raha, D. Kulkarni, R. Shorey, J. Tew, and V. Raghunathan, "D-PUF: An Intrinsically Reconfigurable DRAM PUF for Device Authentication and Random Number Generation," *ACM Transactions on Embedded Computing Systems*, vol. 17, no. 1, pp. 17:1–17:31, Dec. 2017. [Online]. Available: http://doi.acm.org/10.1145/3105915

[9] S. Sutar, A. Raha, and V. Raghunathan, "Memory-Based Combination PUFs for Device Authentication in Embedded Systems," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 4, no. 4, pp. 793–810, Oct 2018.

[10] S. Sutar and V. Raghunathan, "A Lightweight End-to-End Authentication Protocol for Implantable Medical Devices," *Under Review.*

[11] C. Keller, F. Gürkaynak, H. Kaeslin, and N. Felber, "Dynamic memory-based physically unclonable function for the generation of unique identifiers and true random numbers," in *International Symposium on Circuits and Systems (ISCAS)*, June 2014, pp. 2740–2743.

[12] M. S. Hashemian, B. Singh, F. Wolff, D. Weyer, S. Clay, and C. Papachristou, "A robust authentication methodology using physically unclonable functions in DRAM arrays," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2015, pp. 647–652.

[13] F. Tehranipoor, N. Karimian, K. Xiao, and J. Chandy, "DRAM based Intrinsic Physical Unclonable Functions for System Level Security," in *Great Lakes Symposium on VLSI*, 2015, pp. 15–20.

[14] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber, "Modeling Attacks on Physical Unclonable Functions," in *Conference on Computer and Communications Security (CCS)*. New York, NY, USA: ACM, 2010, pp. 237–249. [Online]. Available: http://doi.acm.org/10.1145/1866307.1866335

[15] P. T. Tuyls, G. J. Schrijen, and D. W. E. Schobben, "Distributed PUF," 2014, Intrinsic ID B.V., US Patent 8,699,714.

[16] M. Wang, A. Yates, and I. L. Markov, "SuperPUF: Integrating heterogeneous Physically Unclonable Functions," in *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2014, pp. 454–461.

[17] Transparency Market Research, "Implantable Medical Devices Market: Global Industry Analysis, Size, Share, Growth, Trends, and Forecast 2016-2024," 2016. [Online]. Available: https://www.transparencymarketresearch.com/global-implantable-medical-devices-market.html

[18] Medtronic AZURE Pacing System, *https://www.medtronic.com/us-en/healthcare-professionals/products/cardiac-rhythm/pacemakers/azure.html.*

[19] D. Halperin, T. S. Heydt-Benjamin, B. Ransford, S. S. Clark, B. Defend, W. Morgan, K. Fu, T. Kohno, and W. H. Maisel, "Pacemakers and Implantable Cardiac Defibrillators: Software Radio Attacks and Zero-Power Defenses," in *IEEE S&P*, 2008.

[20] M. Rushanan, A. D. Rubin, D. F. Kune, and C. M. Swanson, "SoK: Security and Privacy in Implantable Medical Devices and Body Area Networks," in *IEEE S&P*, 2014.

[21] A. Bogdanov, D. Khovratovich, and C. Rechberger, "Biclique Cryptanalysis of the Full AES," in *ASIACRYPT*, 2011.

[22] M. Zhang, A. Raghunathan, and N. K. Jha, "Towards trustworthy medical devices and body area networks," in *2013 DAC*, May 2013, pp. 1–6.

[23] M. Rostami, W. Burleson, F. Koushanfar, and A. Juels, "Balancing Security and Utility in Medical Devices?" in *DAC*, 2013. [Online]. Available: http://doi.acm.org.ezproxy.lib.purdue.edu/10.1145/2463209.2488750

[24] K. B. Rasmussen, C. Castelluccia, T. S. Heydt-Benjamin, and S. Capkun, "Proximity-based Access Control for Implantable Medical Devices," in *ACM CCS*, 2009. [Online]. Available: http://doi.acm.org/10.1145/1653662.1653712

[25] M. Rostami, A. Juels, and F. Koushanfar, "Heart-to-heart (H2H): Authentication for Implanted Medical Devices," in *ACM CCS*, 2013. [Online]. Available: http://doi.acm.org/10.1145/2508859.2516658

[26] F. Xu, Z. Qin, C. C. Tan, B. Wang, and Q. Li, "IMDGuard: Securing implantable medical devices with the external wearable guardian," in *IEEE INFOCOM*, 2011.

[27] Y. Kim, W. S. Lee, V. Raghunathan, N. K. Jha, and A. Raghunathan, "Vibration-based secure side channel for medical devices," in *DAC*, 2015.

[28] C. S. Jang, D. G. Lee, J. Han, and J. H. Park, "Hybrid Security Protocol for Wireless Body Area Networks," *Wireless Communication & Mobile Computing*, 2011. [Online]. Available: http://dx.doi.org/10.1002/wcm.884

[29] T. Xu, J. B. Wendt, and M. Potkonjak, "Matched Digital PUFs for Low Power Security in Implantable Medical Devices," in *IEEE International Conference on Healthcare Informatics*, 2014, pp. 33–38.

[30] C. Fu, X. Du, L. Wu, Q. Zeng, A. Mohamed, and M. Guizani, "POKs Based Secure and Energy-Efficient Access Control for Implantable Medical Devices," in *Security and Privacy in Communication Networks.* Cham: Springer International Publishing, 2019, pp. 105–125.

[31] J. Delvaux, R. Peeters, D. Gu, and I. Verbauwhede, "A Survey on Lightweight Entity Authentication with Strong PUFs," *ACM Comput. Surv.*, vol. 48, no. 2, pp. 26:1–26:42, Oct. 2015. [Online]. Available: http://doi.acm.org/10.1145/2818186

[32] U. Chatterjee, V. Govindan, R. Sadhukhan, D. Mukhopadhyay, R. S. Chakraborty, D. Mahata, and M. M. Prabhu, "Building PUF Based Authentication and Key Exchange Protocol for IoT Without Explicit CRPs in Verifier Database," *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 3, pp. 424–437, May 2019.

[33] G. E. Suh and S. Devadas, "Physical Unclonable Functions for Device Authentication and Secret Key Generation," in *Design Automation Conference (DAC).* New York, NY, USA: ACM, 2007, pp. 9–14. [Online]. Available: http://doi.acm.org/10.1145/1278480.1278484

[34] R. Pappu, B. Recht, J. Taylor, and N. Gershenfeld, "Physical One-Way Functions," *Science*, vol. 297, no. 5589, pp. 2026–2030, 2002. [Online]. Available: http://science.sciencemag.org/content/297/5589/2026

[35] S. T. C. Konigsmark, D. Chen, and M. D. F. Wong, "PolyPUF: Physically Secure Self-Divergence," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 7, pp. 1053–1066, July 2016.

[36] A. Raha, S. Sutar, H. Jayakumar, and V. Raghunathan, "Quality Configurable Approximate DRAM," *IEEE Transactions on Computers*, vol. 66, no. 7, pp. 1172–1187, July 2017.

[37] A. Raha, H. Jayakumar, S. Sutar, and V. Raghunathan, "Quality-aware data allocation in approximate DRAM*," in *2015 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, Oct 2015, pp. 89–98.

[38] W. Xiong, A. Schaller, N. Anagnostopoulos, M. U. Saleem, S. Gabmeyer, S. Katzenbeisser, and J. Szefer, "Run-time Accessible DRAM PUFs in Commodity Devices," in *Conference on Cryptographic Hardware and Embedded Systems (CHES)*, May 2016.

[39] F. Tehranipoor, W. Yan, and J. A. Chandy, "Robust hardware true random number generators using DRAM remanence effects," in *International Symposium on Hardware Oriented Security and Trust (HOST)*, May 2016, pp. 79–84.

[40] F. Tehranipoor, N. Karimian, W. Yan, and J. A. Chandy, "DRAM-Based Intrinsic Physically Unclonable Functions for System-Level Security and Authentication," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 3, pp. 1085–1097, March 2017.

[41] J. S. Kim, M. Patel, H. Hassan, and O. Mutlu, "The DRAM Latency PUF: Quickly Evaluating Physical Unclonable Functions by Exploiting the Latency-Reliability Tradeoff in Modern Commodity DRAM Devices," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2018, pp. 194–207.

[42] N. A. Anagnostopoulos, S. Katzenbeisser, J. Chandy, and F. Tehranipoor, "An Overview of DRAM-Based Security Primitives," *Cryptography*, vol. 2, no. 2, 2018. [Online]. Available: http://www.mdpi.com/2410-387X/2/2/7

[43] D. E. Holcomb, W. P. Burleson, and K. Fu, "Power-Up SRAM State as an Identifying Fingerprint and Source of True Random Numbers," *IEEE Transactions on Computers*, vol. 58, no. 9, pp. 1198–1210, Sept 2009.

[44] C. Böhm and M. Hofer and W. Pribyl, "A microcontroller SRAM-PUF," in *2011 5th International Conference on Network and System Security*, Sept 2011, pp. 269–273.

[45] A. Bacha and R. Teodorescu, "Authenticache: Harnessing cache ECC for system authentication," in *International Symposium on Microarchitecture (MICRO)*, Dec 2015, pp. 128–140.

[46] F. Zhang, S. Yang, J. Plusquellic, and S. Bhunia, "Current based PUF exploiting random variations in SRAM cells," in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2016, pp. 277–280.

[47] A. R. Krishna, S. Narasimhan, X. Wang, and S. Bhunia, "MECCA: A Robust Low-Overhead PUF Using Embedded Memory Array," in *Cryptographic Hardware and Embedded Systems – CHES 2011*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 407–42.

[48] D. E. Holcomb and K. Fu, "Bitline PUF: Building Native Challenge-Response PUF Capability into Any SRAM," in *Proceedings of the 16th International Workshop on Cryptographic Hardware and Embedded Systems — CHES 2014 - Volume 8731*. Berlin, Heidelberg: Springer-Verlag, 2014, pp. 510–526. [Online]. Available: https://doi.org/10.1007/978-3-662-44709-3_28

[49] P. Prabhu, A. Akel, L. M. Grupp, W. S. Yu, G. E. Suh, E. Kan, and S. Swanson, “Extracting Device Fingerprints from Flash Memory by Exploiting Physical Variations,” in *Proceedings of the 4th International Conference on Trust and Trustworthy Computing*, ser. TRUST’11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 188–201. [Online]. Available: http://dl.acm.org/citation.cfm?id=2022245.2022264

[50] Y. Wang, W. K. Yu, S. Wu, G. Malysa, G. E. Suh, and E. C. Kan, “Flash Memory for Ubiquitous Hardware Security Functions: True Random Number Generation and Device Fingerprints,” in *2012 IEEE Symposium on Security and Privacy*, May 2012, pp. 33–47.

[51] W. Che, J. Plusquellic, and S. Bhunia, “A non-volatile memory based physically unclonable function without helper data,” in *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2014, pp. 148–153.

[52] D. Lim, “Extracting Secret Keys from Integrated Circuits,” in *Master’s thesis, Massachusetts Institute of Technology*, May 2004.

[53] K. Kursawe, A. R. Sadeghi, D. Schellekens, B. Skoric, and P. Tuyls, “Reconfigurable Physical Unclonable Functions - Enabling technology for tamper-resistant storage,” in *International Workshop on Hardware-Oriented Security and Trust (HOST)*, July 2009, pp. 22–29.

[54] I. Eichhorn, P. Koeberl, and V. van der Leest, “Logically Reconfigurable PUFs: Memory-based Secure Key Storage,” in *Workshop on Scalable Trusted Computing (STC)*. New York, NY, USA: ACM, 2011, pp. 59–64. [Online]. Available: http://doi.acm.org/10.1145/2046582.2046594

[55] C. Herder, M. D. Yu, F. Koushanfar, and S. Devadas, “Physical Unclonable Functions and Applications: A Tutorial,” *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1126–1141, Aug 2014.

[56] H. Jayakumar, A. Raha, Y. Kim, S. Sutar, W. S. Lee, and V. Raghunathan, “Energy-efficient system design for IoT devices,” in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan 2016, pp. 298–301.

[57] J. Liu, B. Jaiyen, Y. Kim, C. Wilkerson, and O. Mutlu, “An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms,” in *International Symposium on Computer Architecture (ISCA)*. New York, NY, USA: ACM, 2013, pp. 60–71. [Online]. Available: http://doi.acm.org/10.1145/2485922.2485928

[58] W. Che, F. Saqib, and J. Plusquellic, “PUF-Based Authentication,” in *International Conference on Computer-Aided Design (ICCAD)*. Piscataway, NJ, USA: IEEE Press, 2015, pp. 337–344. [Online]. Available: http://dl.acm.org/citation.cfm?id=2840819.2840867

[59] U. Kocabaş, A. Peter, S. Katzenbeisser, and A. Sadeghi, “Converse PUF-Based Authentication,” in *International Conference on Trust and Trustworthy Computing (TRUST)*. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 142–158. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-30921-2_9

[60] P. Restle, J. Park, and B. Lloyd, “DRAM variable retention time,” in *International Technical Digest on Electron Devices Meeting*, 1992.

[61] M. K. Qureshi, D. H. Kim, S. Khan, P. J. Nair, and O. Mutlu, "AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems," in *International Conference on Dependable Systems and Networks*, June 2015, pp. 427–437.

[62] D. S. Yaney, C. Y. Lu, R. A. Kohler, M. J. Kelly, and J. T. Nelson, "A meta-stable leakage phenomenon in DRAM charge storage - Variable hold time," in *International Electron Devices Meeting*, Dec 1987, pp. 336–339.

[63] M. Chang, J. Lin, C. Lai, R. Chang, S. N. Shih, M. Wang, and P. I. Lee, "Si-H bond breaking induced retention degradation during packaging process of 256 mbit DRAMs with negative wordline bias," *IEEE Transactions on Electron Devices*, vol. 52, no. 4, pp. 484–491, April 2005.

[64] H. Kim, B. Oh, Y. Son, K. Kim, S. Y. Cha, J. G. Jeong, S. J. Hong, and H. Shin, "Characterization of the Variable Retention Time in Dynamic Random Access Memory," *IEEE Transactions on Electron Devices*, vol. 58, no. 9, pp. 2952–2958, Sept 2011.

[65] Terasic, "TR4 FPGA development kit," March 2015. [Online]. Available: http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=683

[66] Altera, "Nios II processor," March 2015. [Online]. Available: https://www.altera.com/products/processors/overview.html

[67] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn, "Flikker: Saving DRAM Refresh-power Through Critical Data Partitioning," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. New York, NY, USA: ACM, 2011, pp. 213–224. [Online]. Available: http://doi.acm.org/10.1145/1950365.1950391

[68] JEDEC, "Acceleration factor, temperature," 2016. [Online]. Available: https://www.jedec.org/standards-documents/dictionary/terms/acceleration-factor-temperature

[69] F. Jensen, "Activation energies and the arrhenius equation," *Quality and Reliability Engineering International*, vol. 1, no. 1, pp. 13–17, 1985.

[70] L. E. Bassham, III, A. L. Rukhin, J. Soto, J. R. Nechvatal, M. E. Smid, E. B. Barker, S. D. Leigh, M. Levenson, M. Vangel, D. L. Banks, N. A. Heckert, J. F. Dray, and S. Vo, "SP 800-22 Rev. 1a. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications," Gaithersburg, MD, United States, 2010.

[71] P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," in *Advances in Cryptology — CRYPTO' 99*, Berlin, Heidelberg, 1999, pp. 388–397.

[72] R. van den Berg, B. Skoric, and V. van der Leest, "Bias-based Modeling and Entropy Analysis of PUFs," in *Proceedings of the 3rd International Workshop on Trustworthy Embedded Devices*, ser. TrustED '13. New York, NY, USA: ACM, 2013, pp. 13–20. [Online]. Available: http://doi.acm.org/10.1145/2517300.2517301

[73] C. Helfmeier, C. Boit, D. Nedospasov, and J. P. Seifert, "Cloning physically unclonable functions," in *2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, June 2013, pp. 1–6.

[74] S. Katzenbeisser, U. Kocabaş, V. Rožić, A.-R. Sadeghi, I. Verbauwhede, and C. Wachsmann, "PUFs: Myth, Fact or Busted? A Security Evaluation of Physically Unclonable Functions (PUFs) Cast in Silicon," in *Proceedings of the 14th International Conference on Cryptographic Hardware and Embedded Systems*, ser. CHES'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 283–301. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-33027-8_17

[75] A. Maiti and P. Schaumont, "The Impact of Aging on a Physical Unclonable Function," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 9, pp. 1854–1864, Sept 2014.

[76] K. K. Chang, A. G. Yaglikci, S. Ghose, A. Agrawal, N. Chatterjee, A. Kashyap, D. Lee, M. O'Connor, H. Hassan, and O. Mutlu, "Understanding Reduced-Voltage Operation in Modern DRAM Devices: Experimental Characterization, Analysis, and Mechanisms," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 1, no. 1, pp. 10:1–10:42, Jun. 2017. [Online]. Available: http://doi.acm.org/10.1145/3084447

[77] SK hynix, "DDR3 SDRAM." [Online]. Available: https://www.skhynix.com/products.do?lang=eng&ct1=36&ct2=38&rc=com

[78] EFM32 Tiny Gecko 11 Starter Kit, *https://www.silabs.com/products/development-tools/mcu/32-bit/efm32-tiny-gecko-tg11-starter-kit*. [Online]. Available: https://www.silabs.com/products/development-tools/mcu/32-bit/efm32-tiny-gecko-tg11-starter-kit

[79] Wireless Xpress BGX13P Starter Kit, *https://www.silabs.com/products/development-tools/wireless/bluetooth/bgx13p-bluetooth-xpress-starter-kit*. [Online]. Available: https://www.silabs.com/products/development-tools/wireless/bluetooth/bgx13p-bluetooth-xpress-starter-kit

[80] Raspberry Pi 3 Model B+, *https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/*. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/

[81] Multi-Node Energy Profiler, *https://www.silabs.com/documents/public/user-guides/ug343-multinode-energy-profiler.pdf*. [Online]. Available: https://www.silabs.com/documents/public/user-guides/ug343-multinode-energy-profiler.pdf

[82] M. T. Rahman, D. Forte, Xiaoxiao Wang, and M. Tehranipoor, "Enhancing noise sensitivity of embedded SRAMs for robust true random number generation in SoCs," in *2016 IEEE Asian Hardware-Oriented Security and Trust (AsianHOST)*, Dec 2016, pp. 1–6.

[83] F. Zhang, S. Yang, J. Plusquellic, and S. Bhunia, "Current based PUF exploiting random variations in SRAM cells," in *2016 DATE*, March 2016, pp. 277–280.

[84] Y. Wang, W. K. Yu, S. Wu, G. Malysa, G. E. Suh, and E. C. Kan, "Flash Memory for Ubiquitous Hardware Security Functions: True Random Number Generation and Device Fingerprints," in *IEEE S&P*, 2012.

[85] D. E. Holcomb, W. P. Burleson, and K. Fu, “Power-Up SRAM State as an Identifying Fingerprint and Source of True Random Numbers,” *IEEE Transactions on Computers*, 2009.

[86] S. Katzenbeisser, U. Kocabaş, V. Rožić, A. Sadeghi, I. Verbauwhede, and C. Wachsmann, “PUFs: Myth, Fact or Busted? A Security Evaluation of Physically Unclonable Functions (PUFs) Cast in Silicon,” in *CHES*, 2012. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-33027-8_17

VITA

VITA

Soubhagya Sutar received the B.Tech. degree in Electronics and Instrumentation Engineering from the National Institute of Technology (NIT), Rourkela, India, in 2011. After working as a control and automation engineer with one of the leading electric utility companies in India, he joined the Direct Ph.D. program in the School of Electrical and Computer Engineering at Purdue University, USA, in 2014.

Mr. Sutar's research interests lie in the design of hardware and software architectures for embedded systems, secure system design, hardware security, and the Internet-of-Things. As a graduate research intern, he worked at the TCS Innovation Labs, Cincinnati, USA and Intel's Microarchitecture Research Lab, Bangalore, India in 2016 and 2017, respectively. He also interned with AMD's Firmware Engineering group at Austin, USA in 2019.

Mr. Sutar is a recipient of the Ross Fellowship from Purdue University in 2014, the Institute Silver Medal from NIT Rourkela in 2011, and the Summer Research Fellowship from the Indian Academy of Sciences in 2009. He received a Best Paper nomination at the International Conference on Compilers, Architectures, and Synthesis for Embedded Systems (CASES) in 2016.