University of Sheffield

Leveraging DRAM-based Physically Unclonable Functions for Enhancing Authentication in Resource-Constrained Applications

Owen Joseph William Millwood

Supervisors: Dr. Prosanta Gope & Dr. Chenghua Lin

This dissertation is submitted
for the degree of *Doctor of Philosophy*

in the

Department of Computer Science
University of Sheffield

October 9th, 2023

Declaration

I declare that this thesis is a presentation of original work and I am the sole author. Work submitted for this research degree at the University of Sheffield has not formed part of any other degree either at the University of Sheffield or at another establishment. All sources are acknowledged as references. Certain parts of the material presented within this thesis have appeared in published or submitted papers elsewhere. Specifically, these are:

- **Millwood, O.**, Miskelly, J., Yang, B., Gope, P., Kavun, E. B., & Lin, C. (2023). PUF-Phenotype: A Robust and Noise-Resilient Approach to Aid Group-Based Authentication With DRAM-PUFs Using Machine Learning. IEEE Transactions on Information Forensics and Security (TIFS).
- **Millwood, O.**, Pehlivanoglu, M. K., Pasikhani, A. M., Miskelly, J., Gope, P., & Kavun, E. B. (2023). A Generic Obfuscation Framework for Preventing ML-Attacks on Strong-PUFs through Exploitation of DRAM-PUFs. 8th IEEE European Symposium on Security and Privacy (EuroS&P).
- **Millwood, O.**, Hongming, F., Gope, P., Narlı, O., Pehlivanoğlu, M. K., Kavun, E. B., & Sikdar, B. (2023). A Privacy-Preserving Protocol Level Approach to Prevent Machine Learning Modelling Attacks on PUFs in the Presence of Semi-Honest Verifiers. 2023 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), 326–336. IEEE.

Name: Owen Joseph William Millwood

Signature:

Date: September 26, 2024

I dedicate this thesis to Gráinne, Tony, Robert and Caitlin.
For believing in me when I could not, this is woven with the fabric of your support.

Abstract

The Internet-of-Things (IoT) is an inevitable technological paradigm shift across hundreds of application domains, including healthcare, manufacturing, and infrastructure. Communication of potentially sensitive or security-critical information is commonplace as with any technological domain. Therefore, the security of such communication is key to providing trust to enable wider adoption. IoT as a concept has many implications that need not always be assumed for other communication environments, such as physical access and power/hardware restrictions, which present challenges to established methods for securing communication. This reality means that assumptions regarding the storage and transmission of sensitive data must be stronger. In response to the new demands for such secure communication, Physical Unclonable Functions (PUFs) were proposed to provide highly secure just-in-time cryptographic tokens/keys to enable strong security at a low resource overhead. Various types of PUF have been proposed since their conception, each with varying features, strengths and weaknesses. Most notably, PUFs proposed have been consistently subjected to machine learning modelling attacks (ML-MA). Dynamic Random Access Memory (DRAM) PUFs were more recently proposed with promising features over similar PUF types. Limited works have been explored investigating the applicability of utilising DRAM-PUFs to design secure authentication protocols. Therefore, it is essential to develop techniques for and test the effectiveness of DRAM-PUFs for enabling authentication in resource-constrained systems.

This thesis proposes three novel methods for enabling authentication by exploiting DRAM-PUFs. Particularly, we develop a computer vision-based approach for accurately classifying and retrieving noisy Latency DRAM-PUF responses for grouped devices, noting the highest-performing classifiers for this task. Through this research, we develop and share a unique temperature and voltage-dependent Latency DRAM-PUF dataset for use of the wider research community. To improve the authentication scalability, we additionally present a novel, generic obfuscation method for Strong PUFs, further demonstrating how utilisation of DRAM-PUF characteristics can ensure high security against machine learning threats. Finally, a hardware-level approach is tightly integrated into a privacy-preserving authentication protocol to enhance protection against threats with advanced prior PUF access and knowledge to prevent powerful ML-MA.

Acknowledgements

I will first profoundly thank my primary supervisor, Dr Prosanta Gope, for whom without, this opportunity and thesis would simply not have been possible. His support, expertise, patience (especially) and guidance has been consistently above and beyond the call of duty.

I also deeply thank my other two supervisors. Firstly, Dr. Chenghua Lin, who provided invaluable insights into many aspects of my experimentation and writing process, helping me grow as a scientist. Secondly, Professor Elif Bilge Kavun, who continued to make time to be an extremely active and supportive supervisor and mentor to me where she had no obligation to after moving to a new university.

Next, I would like to massively thank Dr. Jack Miskelly, Dr. Meltem Kurt Pehlivanoglu, Bohao Yang, Oğuz Narlı, Dr. Biplap Sikdar and Ping-Chen Lin, who I have had the pleasure to work with, publish with and learn from during my studies. Each person has helped with carrying out experiments, providing expertise and writing up for our successful publications.

I give a special thanks to my colleague and friend Dr. Aryan Pasikhani, who became a big brother to me from my first week at the university and has always been there with a big smile ready to offer me guidance and support. Also, Alexandra Herghelegiu, for often snapping me into gear when my ideas got the better of me, and for just being very clever in general by helping give birth to the concept of a PUF Phenotype. A big thank you goes to the rest of my friends and colleagues at the University of Sheffield, who helped make the journey so enjoyable.

Finally, my mother, Gráinne, father Robert, step-father Tony and sister Caitlin have my never-ending gratitude for their unconditional love and support over the past four years; Their almost unreasonable amount of belief in me kept me moving one foot in front of the other every step of the way.

This process has been a culmination of growth not possible without all the names mentioned here, who were the big fish in the big pond I found myself in, and I attribute so much of my current knowledge and skills to what I have learned from each of you over the past four years. Thank you.

Grants The work in this thesis was supported by The Royal Society Research Grant under grant RGS\R1\221183 as well as the National Research Foundation, Singapore, under grant NRF2018NCRNCR002-0001.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The need to prove the identity and authenticity of objects and communications has been an important problem to be solved for thousands of years. Evidence for using signatures to identify and authenticate objects even dates back as early as 3100 BC by the Sumerians of Mesopotamia [35]. In the modern world of electronic technology, the requirement for deriving strong identity and authenticity has remained of paramount importance. As society primarily adopts computerised technologies for storing, processing and communicating potentially private, sensitive or even safety-critical data, providing strong trust becomes a significant problem to tackle. While many methods have been developed to enable various levels of trust in digital communications (for example, Transport Layer Security [18]), often, most security assumptions are based on the cryptographic assumptions of the ciphers and software which underpin the protocols. With the ever-growing adoption of technologies such as the Internet-of-Things (IoT), increasingly, smart components and devices must communicate openly on publicly accessible networks with fewer computational and power resources available and - arguably most significantly - generally must be assumed to be physically accessible to potential adversaries. With these new assumptions, novel problems arise in protecting the security and practicality of communications. The concept of a 'Root-of-Trust' (RoT) describes the lowest level of security in the hardware/software abstraction stack* of which where trust is assumed, further trust may be built upon it. While each layer must include its own security architecture to prevent an attack at the given level, security vulnerabilities in lower abstractions cause security measures at higher abstractions to become redundant as they can then be circumvented. Assuming attackers have physical access to devices when considering the domain of IoT, physical hardware attacks must be considered. Suddenly, hardware-level RoTs become essential to enable the overall security of the entire abstraction stack. If the hardware cannot be trusted, the software cannot be trusted.

Physically Unclonable Functions (PUFs) were proposed as a method to provide a physical RoT for electronic devices by exploiting entropy present in the integrated circuits (ICs) of which a device is comprised [27]. Through the characterisation of micro-variations which occur during the manufacturing process of the IC, PUFs enable system designers to provide strong authentication and even key generation for embedded security applications based upon an unpredictable and unique feature of an individual device. Since their conception,

*Application code (highest-level software) down to the physics of the hardware itself, with levels of software such as assembly code, machine code, instruction set architecture etc. in between.

many different PUF designs have been proposed, each with various features, providing unique strengths and weaknesses for enabling security applications.

Certain PUFs known as 'Strong'/'Extensive' PUFs (described in detail in Section 2.3.1) showed promising properties for use in ultra-lightweight challenge/response authentication protocols due to an ability to produce an exponentially growing number of unique input/output pairs - known as challenge/response pairs (CRPs) - with physical PUF size. Verifiers could send one-time challenges to PUF devices with an expected response (already stored in a database after an initial enrollment period), and the PUF device could excite the PUF to generate the response on the fly and return it to the verifier in order to perform authentication. Shortly after this proposal, however, it was discovered that it is possible to utilise machine learning algorithms to clone PUFs using a relatively small amount of CRPs from the total CRP space, enabling a counterfeit model to accurately output correct responses to novel challenges, effectively allowing attackers to impersonate the PUF, and thus the PUF device [74]. Since then, all strong PUF proposals have been later shown to be vulnerable to machine learning modelling attacks (ML-MA) [16, 23, 44, 76, 77, 90]. In response to this significant vulnerability, obfuscation schemes have been proposed with the aim to obscure the hardware interface to the underlying PUF circuit and/or reduce the linearity in the correlation between the challenges and responses and thus increase the difficulty for an attacker to perform modelling, physical and side-channel attacks [20,24,28,92,95,96]. Such schemes, however, often either vary in their effectiveness in preventing ML-MA or often require an amount of hardware overhead, which prevents the solution from remaining lightweight enough for reasonable application to a resource-constrained embedded system.

So-called 'Weak'/'Constrained' PUFs are a second key type of PUF which have undergone development concurrently with Strong PUFs, characterised by a single or small number of supported unique CRPs of a large individual size (hence 'Weak', though not a reflection on security properties). These PUFs were originally proposed as relatively complicated optical measurement circuits exploiting unique speckle patterns from shining a laser through a diffuser [71]. Despite the naming convention, Weak PUFs have found broader adoption and applicability within the academic community and have even started to be adopted commercially for key generation purposes as an alternative to key storage in Non-Volatile Memories (NVMs). While the limited CRP space makes Weak PUFs unsuitable for supporting challenge/response-based authentication, it does provide an intrinsic resilience against ML-MA due to a lack of available data to perform sufficiently accurate cloning of the PUF behaviour. Resultantly, some works have also proposed to utilise the stability of Weak PUFs to enhance the security of Strong PUFs in a combined fashion in an attempt to perform obfuscation of Weak PUFs while keeping resource overheads low [14, 53].

Memory PUFs were developed utilising Static and Dynamic Random Access Memories (SRAM/DRAM) to derive entropy from power-cycle/refresh cell-state and read/write latency phenomena. These 'Software PUFs' (denoted by their method of entropy measurement occurring through software) are particularly attractive due to their relative stability and pre-existing presence in computer systems, allowing for efficient use of on-device resources over a requirement for custom hardware additions. While SRAM-PUF has seen extensive exploration in the literature and are one of the very few PUF types which has been adopted commercially, DRAM-PUFs have some key promising features over SRAM, namely, faster PUF access time, higher density (higher available CRP space) and no requirement for a dis-

ruptive device power cycle. Due to the ubiquitous nature of DRAM in computer systems, ranging from high performance to commodity down to some embedded systems, the investigation into methods for exploiting DRAM-PUFs for better enhancing secure applications with advanced threat models becomes an attractive area of research.

PUFs, which measure extremely low-level physical variations, are inherently susceptible to noise induced by environmental factors such as temperature, humidity, voltage fluctuations, and ageing. DRAM-PUFs are no exception to these challenges, meaning measurements often contain noise, which can increase in particularly adverse conditions. Additionally, DRAM consists of memory regions which are typically more stable than others, which is also affected by noise. While unstable regions can be exploited for generating random numbers (given exceptionally unstable characteristics), these regions reduce the effective space for reliable CRP generation suitable for authentication. Furthermore, unstable regions are typically not clustered together physically on the DRAM die itself; rather are scattered at the bit-cell level among the stable cells, making large PUF measurements difficult to remove noise from. In cryptographic applications, noise prevents the reliable generation of secret keys containing zero noise. Commonly, error correction methods are employed, such as Fuzzy Extraction (FE) to remove noise from key data with the aid of helper data; however, they have been shown to leak information which enables ML-MA on the target PUF [79]. Computer-vision techniques have been tested to enable authentication of Memory-PUF responses without performing denoising actively; however, they come with a trade-off in computational overhead.

1.1 Aims and Objectives

DRAM-PUFs demonstrate highly promising characteristics for enabling security in resource-constrained systems. Additionally, there is a significant potential for Strong PUFs to enable highly secure and very lightweight authentication in such systems. Based on the currently unexplored aspects in the current state-of-the-art, we identify the current open questions as hypotheses, with the *first*:

Hypothesis 1: *Computer vision-based authentication can be employed using DRAM-PUFs for authentication in a practically lightweight fashion. Highly acceptable false positive/false negative rates for authentication can be achieved across multiple devices (multi-class classification) for highly noisy responses.*

Since DRAM-PUFs do not support an extremely high number of unique CRPs, it is important to investigate how they can be further integrated alongside currently vulnerable Strong PUFs in order to enable ideal challenge/response-based authentication. Currently, the obfuscation of Strong PUFs is highly resource-consuming, therefore, our *second* hypothesis is:

Hypothesis 2: *DRAM-PUF entropy can be exploited to enhance the obfuscation of Strong PUFs for authentication such that required hardware overhead is minimised while resistance against ML-MA remains high.*

Finally, while obfuscation of Strong PUFs enhances resilience against ML-MA, ensuring authentication protocols are compatible with the design for optimal use remains vital.

Additionally, hardware design alone cannot prevent ML-MA, where adversaries have more advanced knowledge of the PUF at various stages in the device life cycle. We particularly note a *semi-honest* threat, where it is assumed that a previously honest actor in the authentication process becomes dishonest and attempts to exploit previously collected PUF data in order to compromise future authentication, which hardware design alone cannot overcome. For this reason, our *final* hypothesis is as follows:

Hypothesis 3: *A privacy-preserving protocol can be designed which tightly integrates a DRAM-PUF-enhanced Strong-PUF obfuscation design in order to provide further resilience against ML-MA, including resilience against adversaries with significantly elevated privileges to collect PUF data and mount attacks.*

Collectively, the hypotheses (and thus experimental research) attempt to enhance the application of DRAM-PUFs as useful cryptographic primitives for resource-constrained environments, particularly for enabling strong authentication of devices against sophisticated threats.

1.2 Thesis Contributions

As a result of the research carried out based on our initial hypotheses, the major contributions provided in this thesis are as follows:

- The introduction of the concept of a *'PUF Phenotype'* as a way to describe an observed PUF response, including its environmentally-dependant noise features, such that not only underlying PUF structures contribute to identity but also its specific measured interaction with environmental change (i.e. noise).
- The proposal and evaluation of a Convolutional Neural Network (CNN) based authentication methodology for DRAM-PUF 'Phenotypes', enabling accurate identification and authentication of noisy DRAM-PUF responses without a requirement for denoising or error-correction approaches. This approach is performed at a lower overhead than the state-of-the-art and supports group-based authentication through multi-class (multiple device) classification.
- The production of a novel temperature and voltage variant Latency DRAM-PUF response 'Phenotype' dataset.
- The proposal and evaluation of a novel generic and low-cost obfuscation scheme for securing Strong PUFs against ML-MA by exploiting DRAM-PUFs, validated in hardware on Field Programmable Gate Array (FPGA).
- The proposal and evaluation of a novel PUF-based authentication protocol designed to operate with DRAM-PUF-based Strong PUF obfuscating hardware.

1.3 Thesis Overview

The overall thesis is structured as shown in Figure 1.1 to address the aims, objectives and hypotheses described in the earlier subsections. This is presented in the following chapters:

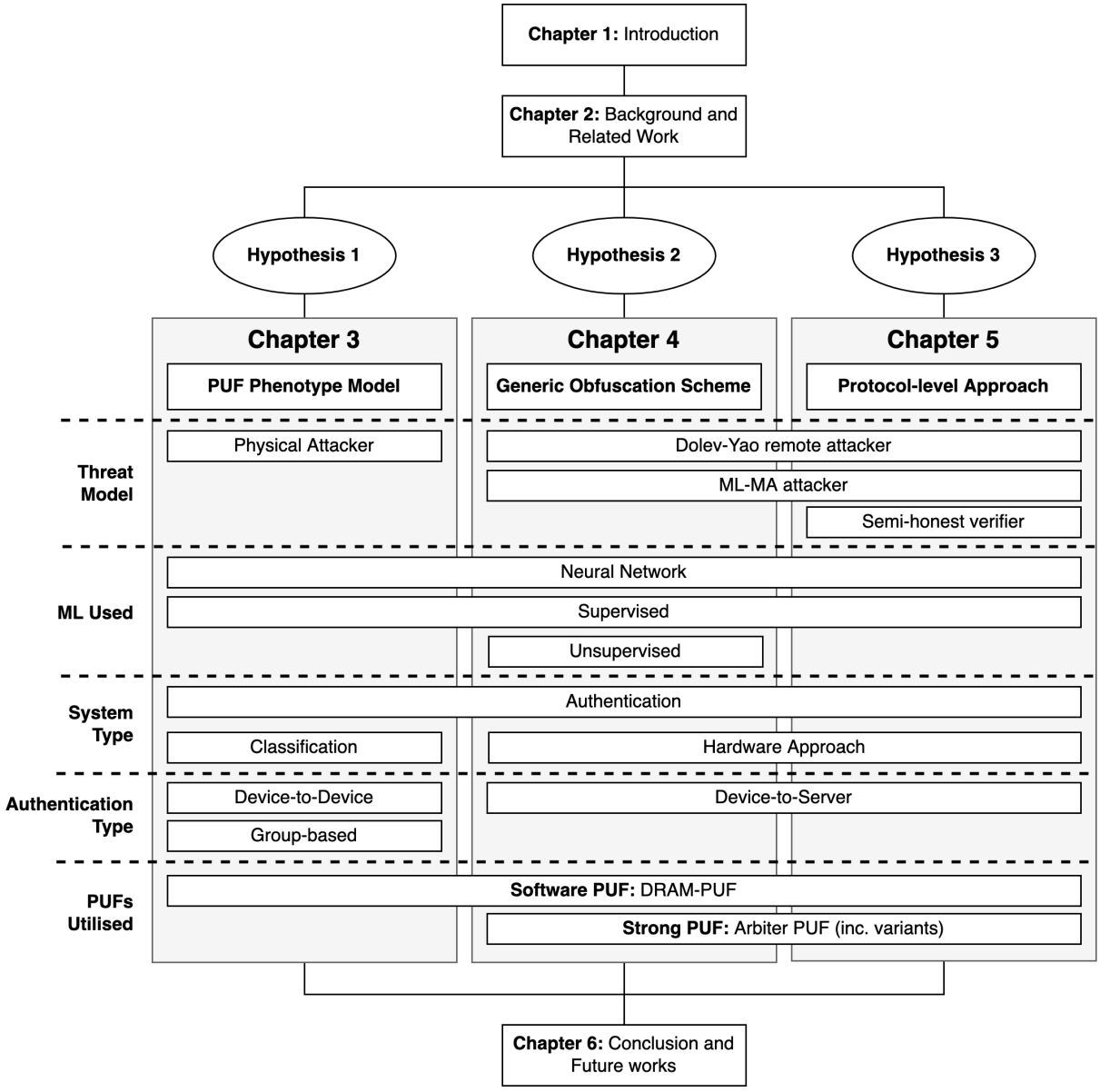

Figure 1.1: *Thesis structure overview*

In **Chapter 2**, titled **'Background and Related Work'**, we provide the definitions of general concepts relevant for the understanding of all technical chapters, including relevant related works, identifying their limitations to supplement the motivations for the undertaking of this research.

In **Chapter 3**, titled **'PUF-Phenotype: A Robust and Noise Resilient Approach to Aid Group-Based Authentication With DRAM-PUFs Using Machine Learning'**, we propose a novel approach to authenticating noisy DRAM-PUF responses. By utilising a combination of CNNs and classical ML classifiers, we classify the origin of a response to a given device in a group and infer the authenticity of the input via model confidence. The proposed approach is empirically tested on several machine learning algorithms, including an ablation study against simulated fraudulent samples.

In **Chapter 4**, titled **'A Generic Obfuscation Framework for Preventing ML-Attacks on Strong PUFs through Exploitation of DRAM-PUFs'**, we introduce a novel approach to obfuscate the challenge/response interface of any Strong PUF. Our method leverages the advantageous traits of DRAM-PUFs to prevent ML-MA when using Strong PUFs for remote authentication. We provide experimental evidence to demonstrate the efficacy of our approach. Furthermore, we implement and evaluate the proposed scheme on a Xilinx Field Programmable Gate Array (FPGA) to assess the associated hardware resource overhead.

In **Chapter 5**, titled **'A Privacy-Preserving Protocol Level Approach to Prevent Machine Learning Modelling Attacks on PUFs in the Presence of Semi-Honest Verifiers'**, we propose an authentication protocol for preventing ML-MA against remote attackers, introducing an additional threat as a 'semi-honest verifier'. We build upon the hardware obfuscation scheme in Chapter 4 and integrate a modified PUF obfuscation, which we synthesise and evaluate on a Xilinx FPGA.

In **Chapter 6**, titled **'Conclusions and Future Work'**, we provide a summary of the significant findings and contributions of the research, concluding with recommendations for future work.

1.4 Key Publications

The works presented in the key technical chapters of this thesis have appeared in the following peer-reviewed publications:

1. **Millwood, O.**, Miskelly, J., Yang, B., Gope, P., Kavun, E. B., & Lin, C. (2023). PUF-Phenotype: A Robust and Noise-Resilient Approach to Aid Group-Based Authentication With DRAM-PUFs Using Machine Learning. IEEE Transactions on Information Forensics and Security (TIFS).
   This publication can be found online at: `https://ieeexplore.ieee.org/document/10099450` *This work is presented in Chapter 3.*

2. **Millwood, O.**, Pehlivanoglu, M. K., Pasikhani, A. M., Miskelly, J., Gope, P., & Kavun, E. B. (2023). A Generic Obfuscation Framework for Preventing ML-Attacks on Strong-PUFs through Exploitation of DRAM-PUFs. 8th IEEE European Symposium on Security and Privacy (EuroS&P).

This publication can be found online at: https://ieeexplore.ieee.org/document/10190495 *This work is presented in Chapter 4*

3. **Millwood, O.**, Hongming, F., Gope, P., Narlı, O., Pehlivanoğlu, M. K., Kavun, E. B., & Sikdar, B. (2023). A Privacy-Preserving Protocol Level Approach to Prevent Machine Learning Modelling Attacks on PUFs in the Presence of Semi-Honest Verifiers. 2023 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), 326–336. IEEE.
   This publication can be found online at: https://ieeexplore.ieee.org/document/10133804 *This work is presented in Chapter 5*

Chapter 2

Background and Related Work

This chapter will present the preliminary knowledge relevant to all components of the thesis, including descriptions of relevant technologies and concepts alongside the relevant state-of-the-art literature.

2.1 Physically Unclonable Functions

A Physically Unclonable Function (PUF) is a hardware-rooted security primitive that is used to provide electronic devices/components with an intrinsic identity, commonly referred to as a 'hardware fingerprint' for a device and was first introduced in silicon in 2002 by Gassend et al. in [27]. A PUF generally comprises a specific electronic circuit which is measurable through some mechanism either internally (intrinsic) or externally (extrinsic) to the containing device and produces unique outputs dependent on sub-atomic micro-variations caused during the manufacturing of the circuit itself. In this way, many PUFs may be built from 'identical' circuits, which provide different characteristics and unique outputs based on this entropy caused during manufacturing. As the individual process variation in each component is highly random, the distribution of variations across all components gives the circuit a unique identity. A PUF, therefore, can fulfil a role similar to a biometric in human authentication systems, with the PUF identity being used to verify the identity of the parent device and/or component [31, 68, 85, 93]. PUFs can also be used as a source of secret information, such as providing the secret key for a cryptographic algorithm without that secret needing to be stored in a non-volatile memory, which could be vulnerable to physical, invasive or side-channel attacks [25,54,80,83]. Unlike traditional cryptographic methods, PUFs aim to exploit unpredictable physical variations present in the hardware itself rather than algorithmically to extract entropy [74]. This enables PUFs to act as a strong hardware Root-of-Trust (RoT), such that the derived secrets are very difficult to extract by an attacker, enabling strong security to built atop of this 'root'.

Theoretically, a PUF may be considered as the function $R = PUF(C)$, which accepts a set of n input values (known as Challenges) $C \in \{c_0, ..., c_n\}$ from which are produced a corresponding output value/set of values (known as Responses) $R \in \{r_0, ..., r_n\}$. Due to the physical nature of a PUF, outputs are generally subject to noise caused by varying environmental conditions (temperature, voltage, humidity, ageing etc.). Resultantly, characterisation and/or post-processing of PUF responses is most often required to ensure the

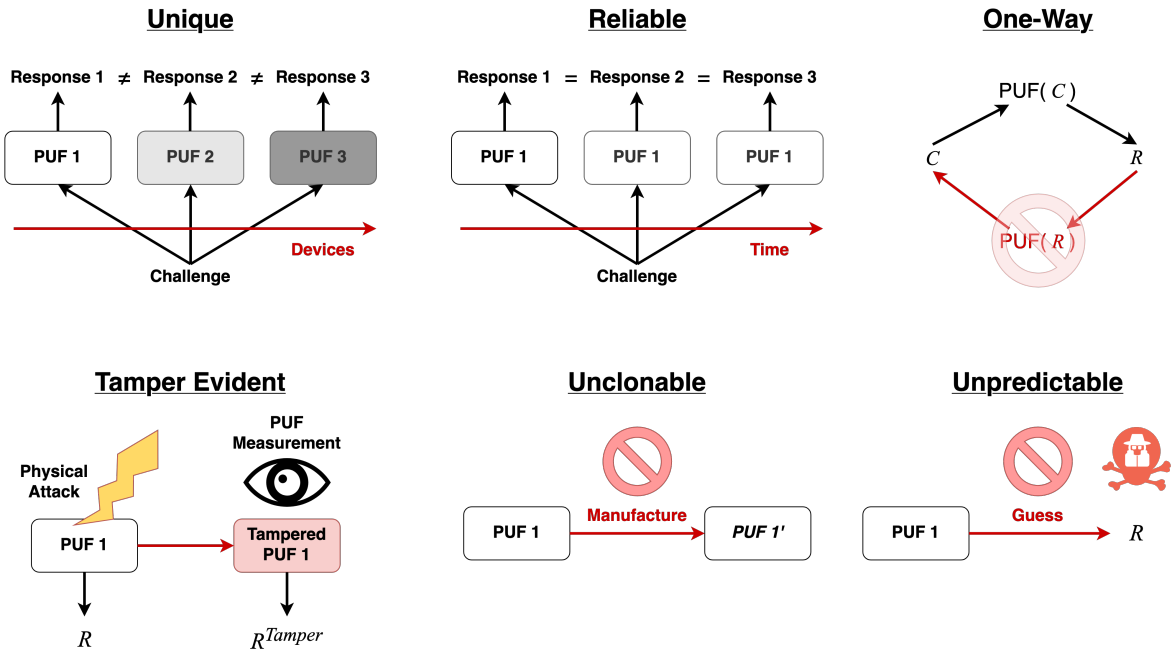

Figure 2.1: *Ideal PUF properties*

reliable derivation of secret keys. Where PUFs are shown to have high instability, however, it has been shown that they can be used effectively as True Random Number Generators (TRNGs) [11, 86].

2.2 Key PUF Properties

In order to evaluate the effectiveness of a PUF design, various key properties must be demonstrated by the PUF and its outputs as illustrated in Figure 2.1. We refer to the definitions provided in [57]:

Uniqueness It must exhibit different challenge-response behaviour. The same challenge issued to n number of PUFs should produce different responses for each.

Reliability/Reproducability The response generated for a given challenge to the same PUF must be consistent through varied use, i.e. the same challenge must produce the same response throughout the lifetime of the PUF, regardless of temporal/environmental variation.

One-Wayness It must be impossible to 'invert' the behaviour of the PUF mathematically, meaning a challenge must not be able to be derived from its arbitrary response.

Tamper-Evidence Attempting to mount a physically invasive attack on a PUF must permanently alter the challenge response behaviour to mitigate its usefulness to an attacker. In addition, this behaviour change will dictate that future legitimate use of the PUF will reveal tampering efforts have occurred.

Unclonability It must be impossible for anyone (including the PUF manufacturer) to construct a new PUF with the identical challenge/response behaviour of another, even if the complete knowledge of another legitimate PUF instance is known.

Unpredictability Knowledge of a given challenge/CRP must not reveal any information about the PUF's response to said challenge. This rule is also considered broken if an adversary can clone a PUF.

2.3 PUF Categories

The concept of a PUF can be realised physically in a variety of ways (see Section 2.4), each resultantly with key varying characteristics which encourage further classification. Most commonly in the literature, PUFs are mainly categorised as 'Weak' (section 2.3.2) or 'Strong' (section 2.3.1). Despite the terminology, this does not refer to the security properties of the PUF design; rather, it reflects the size of the CRP space (the full set of possible unique challenges and responses supported by the PUF)*. Additionally, PUFs can be further divided into unique categories to distinguish their features based on how the PUF itself is measured, denoted as Extrinsic (section 2.3.3) PUFs, Intrinsic PUFs (section 2.3.4) and Software PUFs (section 2.3.5). Strong and Weak categories are mutually exclusive, as well as Extrinsic and Intrinsic categories, otherwise, a PUF can consist of a combination of these categories.

2.3.1 Strong (Extensive) PUFs

Strong PUFs are characterised by a very large CRP space, which ideally grows exponentially with PUF size. A suitable Strong PUF should support sufficient unique CRPs such that a full read-out of all possible CRPs should be highly infeasible for an adversary. Examples of Strong PUFs include Arbiter-based PUFs such as the Arbiter PUF (APUF), XOR-Arbiter PUF (XOR-APUF) and Feed-Forward Arbiter PUF (FF-APUF), discussed in more detail each in the respective Sections 2.4.3, 2.4.4 and 2.4.5. Extremely large CRP spaces make Strong PUFs particularly suited to challenge and response authentication methods as individual CRPs can be used just once (to prevent attackers replaying responses) without concern of the PUF running out of unique CRPs to use (discussed further in Section 2.7). This property of Strong PUFs has been exploited widely, however, in order to perform powerful machine-learning modelling attacks (ML-MA). Due to the significant amount of available CRP data, machine learning models can be trained to accurately model Strong PUFs (discussed in detail in Section 2.5).

2.3.2 Weak (Constrained) PUFs

Weak PUFs –as opposed to Strong PUFs– are characterised by supporting a small CRP space (sometimes only a single CRP), which grows at a generally linear or polynomial rate with hardware size. Weak PUFs are often implemented for key generation purposes as an

*Due to the potentially misleading terminology, the naming convention has been recently updated in the ISO standard: *ISO/IEC 20897-1:2020(E)* to 'Extensive' (Strong) and 'Constrained' (Weak) PUFs. We continue to use the 'Strong' and 'Weak' terminology in this thesis in the meantime as the wider literature adopts these new terms.

alternative to static key storage on NVM. With the limited CRP space, Weak PUFs are intrinsically resilient against ML-MA, resulting in more successful commercial adoption in providing hardware RoT for IoT applications over Strong PUFs. Relevant examples of Weak PUFs include Optical PUFs [71], Static Random Access Memory (SRAM) PUFs [39] (Section 2.4.6) and Dynamic Random Access Memory (DRAM) PUFs [41, 83] (Section 2.4.7).

2.3.3 Extrinsic PUFs

Another useful way of categorising PUFs is based on the evaluation method. Extrinsic PUF(s) measurement of the PUF response requires a method fully external to the PUF itself, for example, the Optical PUF proposed in 2001 by Pappu in [71]. The 'PUF' component of an Optical PUF can be seen as the diffuser material, where the random pattern (and thus entropy) is derived from, and the evaluation method is the optical sensor. Extrinsic PUFs are more challenging to measure as they require external equipment, increasing the hardware overhead for implementation. There is, however, an inherent level of security for Extrinsic PUFs due to the decoupling of the measurement process from the entropy source, increasing the complexity for an attacker.

2.3.4 Intrinsic PUFs

Intrinsic PUFs are PUFs where the evaluation method is integrated into the PUF/device itself, often through the addition of circuitry designed for this purpose. Key examples of Intrinsic PUFs are APUFs, whereby the initial input and output measurement occur in the same circuit [88]. Intrinsic PUFs are extremely useful as often only an interface must be designed to initiate the PUF and read the response directly, for example in the form of a software module. This reduces the resource requirement and complexity of measurements, but may also provide a wider attack surface for adversaries looking to exploit the PUF at a system level.

2.3.5 Software PUFs

Software PUFs are a further important PUF distinction, where the evaluation method is integrated into the PUF and re-purposes *existing* circuitry, using software control mechanisms to evaluate without design changes (thus naturally a subset of Intrinsic PUFs). Where most Intrinsic PUFs at least rely on circuitry designed specifically to be a PUF, Software PUFs use novel manipulations of existing control structures to measure commodity components. This lowers the barrier of entry for using a PUF in a given system and has the desirable property of making the PUF applicable to existing devices without hardware changes. Memory-based PUFs (section 2.4.2) such as SRAM-PUFs, DRAM-PUFs and more recently, Processor-based PUFs [58, 85, 97] are examples of this approach, whereby software controls such as a modified memory controller are utilised to generate and measure PUF entropy.

2.4 PUF Implementations

Since the initial concept of a PUF was first introduced, many different physical implementations for measuring manufacturing variations in electronic devices have been proposed. We

discuss the following PUF implementations relevant to the technical chapters of this thesis, starting with the two relevant key PUF construction distinctions: Delay-based PUFs and Memory PUFs.

2.4.1 Delay-based PUFs

A Delay-based PUF ultimately consists of a type of circuit specifically designed to measure time variation in an input signal travelling through 'identical' circuit pathways [69]. What distinguishes such a circuit as a PUF is that the minute manufacturing variations in these pathways cause input signal travel times which are measurable and different from, circuit to circuit. This grants a given delay circuit a characterisable property which cannot be pre-planned by the manufacturer, that can in turn be exploited as a PUF. A Delay-based PUF can be generally defined by a delay model, which is constituted of the following, as laid out in 2009 by Morozov et al. [69]:

The delay d of a net is defined in 2.1, where d_S is a static delay and d_R is a random delay, caused by manufacturing variations.

$$d = d_S + d_R \tag{2.1}$$

With a Delay-based PUF, consider two nets N_1 and N_2 which must be compared. The delay values for these two nets are defined as:

$$d(N_1) = d_{S1} + d_{R1} \tag{2.2}$$

$$d(N_2) = d_{S2} + d_{R2} \tag{2.3}$$

If the two nets are identical (in circuit layout) then it can be assumed that $d_{S1} = d_{S2}$, therefore, the overall delay Δd between N_1 and N_2 can be expressed as:

$$\Delta d = d_1 - d_2 + d_{R1} - d_{R2} = \Delta d_R \tag{2.4}$$

Ideally, a Delay-based PUF will have a total delay which is solely a function of the random delay component caused by manufacturing variations. If, however, N_1 and N_2 are not laid out identically on the circuit, it is more likely (as it is found in reality) that $d_{S1} \neq d_{S2}$. Because of this, the total delay (skew) becomes:

$$\Delta d = d_{S1} - d_{S2} + d_{R1} - d_{R2} = \Delta d_S + \Delta d_R \tag{2.5}$$

Here, the output of the PUF is at the very least partially influenced by Δd, giving the final output a bias. If Δd_S is ever greater than Δd_R, the effect of the manufacturing is no longer significant, rendering the PUF static, regardless of d_R. For PUF designers therefore, it is important to design a circuit whereby the routing is as identical as manufacturing tolerance allows, formally, such that $\Delta d_S \to 0$.

2.4.2 Memory-based PUFs

Memory-based PUFs are characterised by their composition based upon computational storage hardware, which can be both volatile and non-volatile. Examples include PUFs derived by various phenomena associated with Static, Dynamic and Block Random Access Memories (SRAM, DRAM and BRAM), NAND/NOR flash memory and Read Only Memories (ROM). To remain within scope of the work presented in this thesis, we define PUFs derived from SRAM (section 2.4.6) and DRAM (section 2.4.7).

2.4.3 Arbiter PUF (Strong, Intrinsic, Delay)

The first and arguably most well-documented type of Strong (and Delay-based) PUFs are Arbiter-PUFs (APUFs) – known as a type of 'Linear-Delay PUF' – which exploit intrinsic delay variation in two nominally identical electrical pathways, formalised as an additive linear delay model were introduced by Gassend et al. in 2002 [27].

An APUF consists of k paired multiplexers chained together which determine a pathway an input signal can take through a circuit (Figure 2.2). The output of each multiplexer (and thus the accessible paths) in the chain can be configured by the binary value of the corresponding (to the multiplexers) index of a k bit challenge. Finally, a latch (arbiter) detects the signal and outputs a 0 or 1, depending on which signal was sensed first. Unpredictable manufacturing variations (present in the multiplexers and electrical pathways) ensure that for the same challenge bits as input, different identical APUFs exhibit different outputs, such that each response can uniquely identify a given individual PUF. APUFs, however, are insecure against Machine Learning (ML) based Modelling Attacks (ML-MA), where adversaries who collect sufficient CRPs can accurately determine the final delay value for a given PUF and thus easily output correct responses for unseen challenges [74, 76] (see Section 2.5). Various more comprehensive PUFs based on APUF have been designed and tested in an attempt to reduce the linearity and correlation between the initial challenge data and response data, such as the XOR-Arbiter-PUF (XOR-APUF) (Section 2.4.4).

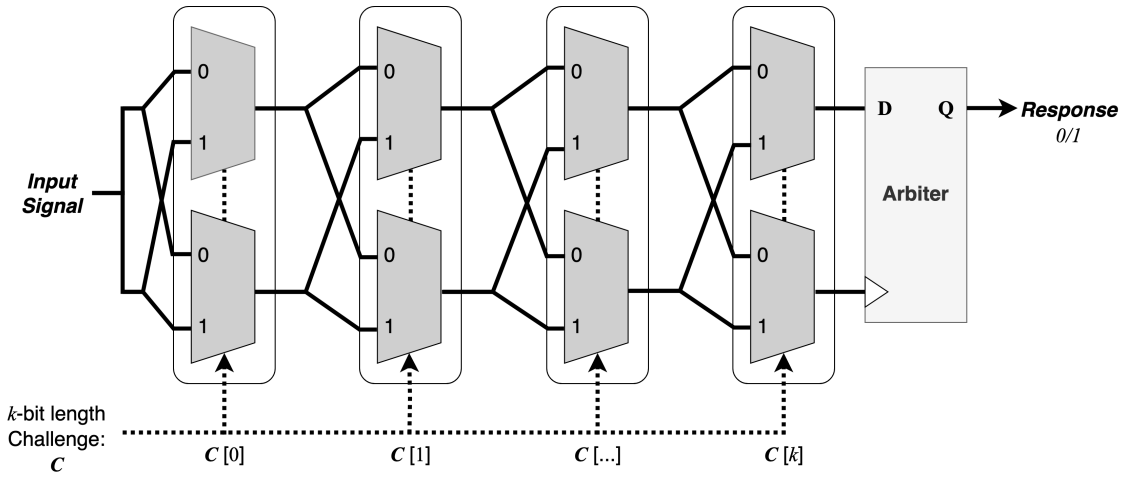

Figure 2.2: *Arbiter-PUF*

2.4.4 XOR-Arbiter PUF (Strong, Intrinsic, Delay)

Due to the linearity (and thus learnability) of the standard APUF, in 2007 Suh et al. suggested using n-APUFs in parallel with each having k-stages to provide non-linearity to the delay model [80]. Figure 2.3 shows the structure of an XOR-APUF, where one unified challenge is applied to each APUF, and each of their different responses is XOR'ed with one another to produce a final 'golden response'. While this XOR logic increases the difficulty for modelling attacks, in 2010 Rührmair et al. proved that machine learning techniques are still able to be utilised to model an XOR-APUF [74] successfully. While it was demonstrated that increasing the number of unique APUFs in the chain increased the resilience against ML-MA, this came at the cost of a significantly increased noise in the overall PUF outputs, requiring more significant error correction to overcome the problem. Including more APUFs in a given IC should also be considered to increase the hardware overhead, which is often undesirable for scalability in the targeted IoT-based solutions.

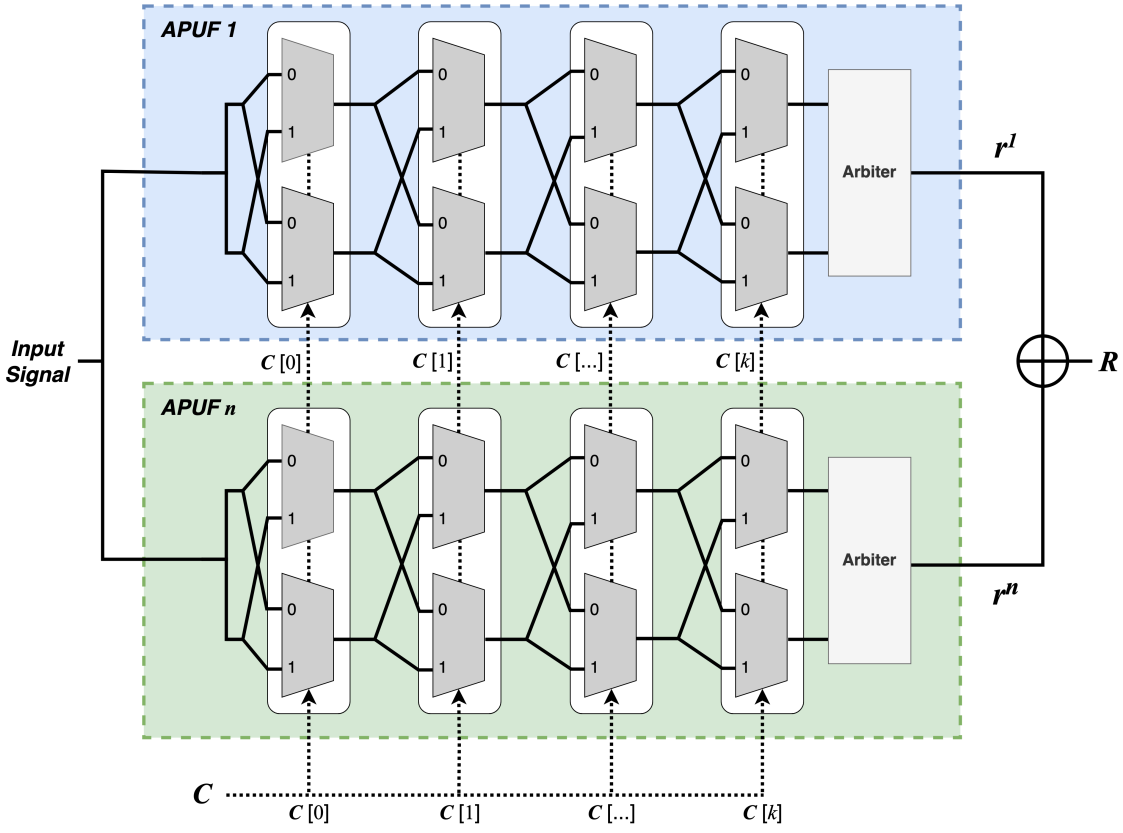

Figure 2.3: *XOR Arbiter-PUF*

2.4.5 Feed-Forward Arbiter PUF (Strong, Intrinsic, Delay)

In 2004, Gassend et al. proposed the Feed-Forward Arbiter PUF (FF-APUF) as another method to reduce the linearity of the standard APUF (Figure 2.4) [29]. In this design, $k - f$ challenge bits are applied to $k - f$ of the multiplexer pairs and the outputs of f pairs are

'fed forward' to multiplexer pairs further in the chain, where f is the number of feed-forward loops included. For each feed-forward loop, the output of both paths is input to an additional arbiter to generate the new challenge bit for the future multiplier pair. While the linearity of the APUF is reduced with the feed-forward mechanism, similar to the XOR-APUF, Gassend et al. raised concern regarding potential vulnerabilities due to the potential to express the FF-APUF in the additive linear delay model [29]. As expected, the FF-APUF was broken by ML-MA by utilising a relatively simple Multi-Layer Perceptron (MLP) by Saed et al. in 2017 [1].

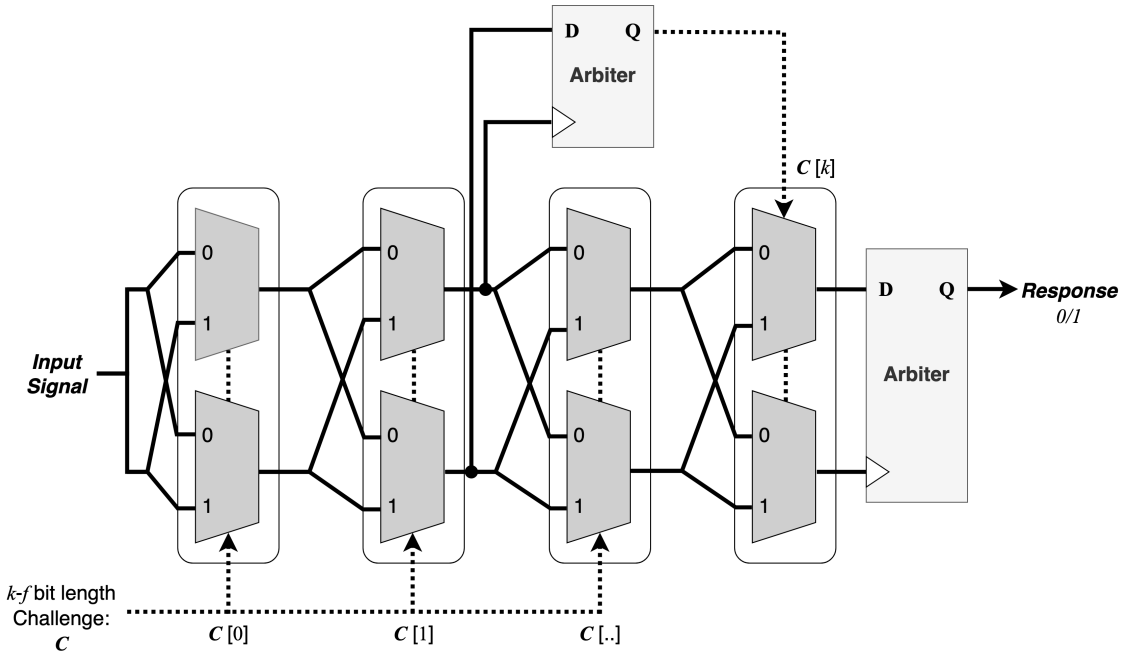

Figure 2.4: *Feed-Forward Arbiter-PUF*

2.4.6 SRAM-PUF (Weak, Intrinsic, Software, Memory)

The SRAM-PUF was first introduced in 2009 by Holcomb et al. in [39]. It exploits unique variations in the bit-cell states on power-up to reveal a digital fingerprint. SRAM stores bits using cross-coupled inverters gated by access transistors (a flip-flop) [72]. A binary *one* or *zero* is stored depending on the state of each side of the inverters, which is denoted as AB: AB = 00 = power down state, AB = 01 = binary zero, AB = 10 = binary one, AB = 11 = unstable and unreachable state. While receiving constant power, an SRAM flip-flop will retain the value stored for an indefinite period of time without requiring any refresh. This property denotes this memory type as being *'Static'* Random Access Memory. As the state of each flip-flop is 00 when no power is supplied, when power is eventually supplied, the state of a given cell has a specific probability of settling into one of the two stable states of binary one or zero. This probability per cell is dictated by manufacturing variation and environmental noise, enabling cells to exhibit cell-state behaviour on start-up, which is unique to the SRAM chip itself and constitutes a PUF. Cells are considered stable and thus suitable for PUF use where the probability of collapsing to one or zero on start-up is high (thus reliable) such that

the same PUF output can be repeatedly obtained. Unstable SRAM regions, found to have a probability of around 0.5 of collapsing to one or zero per bit, while unsuitable for PUF use, are beneficial for use as TRNGs [39]. Figure 2.5 shows how a PUF output is derived from a region of SRAM on start-up. SRAM-PUFs have seen significant research and have even seen some commercial adoption [55, 59].

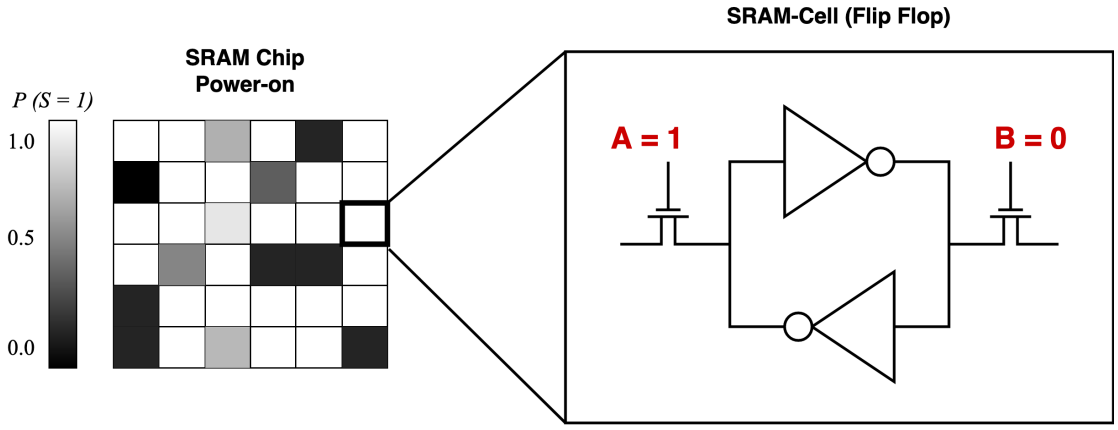

Figure 2.5: *Example of an SRAM-PUF through an SRAM cell within the context of a 36-bit region of SRAM memory during startup, where P is the probability of the cell-state S being a binary one. The stabilised region represents the PUF output.*

2.4.7 DRAM-PUF (Semi-Weak, Intrinsic, Software, Memory)

DRAM-PUFs are a newer form of Memory PUF, yet have certain advantages over SRAM-PUFs which make them desirable for use in resource-constrained environments, such as access time, density and in-runtime availability. Figure 2.6 highlights the organisation of DRAM memory. DRAM consists of a matrix of transistor-gated capacitive cells, each storing a single bit of data, zero or one. Which bit is stored is represented in the charge value of the capacitor in the bit cell. Typically, a fully charged capacitor indicates binary 1, and a fully discharged capacitor indicates binary 0. In practice, this is more complicated at the hardware level due to the existence of what is known as 'anti' and 'true' cells, which store binary zero and one oppositely depending on the value stored in the capacitor; however, it is not relevant for the work carried out in this thesis. Each matrix of cells is referred to as a bank, of which many such banks are organised onto a single DRAM chip.

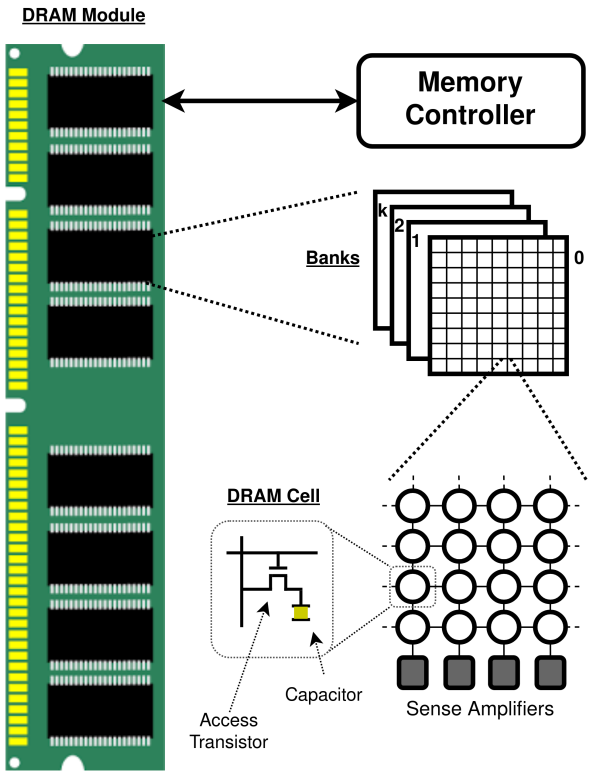

Figure 2.6: *DRAM Organisation [68]*

DRAM cells, unlike SRAM cells, are intrinsically volatile and thus leak charge over time through their capacitors and, given sufficient decay time, can experience a 'bit-flip', whereby the stored contents of the bit-cell changes to the stable state of that particular cell. Therefore, to prevent memory corruption, a periodic refresh cycle must occur to retain stored memory contents during use by reading and writing back cell contents to reset charge values. This property denotes this type of memory as being *'Dynamic'* Random Access Memory. This refresh cycle is typically handled by the memory controller and the time between refresh cycles is known as the Refresh-Pause Interval (RPI).

In 2013, Liu et al. discovered that without the refresh cycle, bit-cells do not fail at a uniform rate. Within the cell arrays is a highly random distribution of cells prone to rapid failure (i.e., highly leaky), a product of process variation in the cell's physical structure [52]. Through this phenomenon, they proposed the first DRAM-PUF, denoted a DRAM 'Retention' PUF, referring to the retention failure process of the bit-cells. As shown in Figure 2.7, the general process is as follows: first, a known pattern (of zeroes and ones) is written into the memory; next, the automatic refresh cycle is halted; the cells are allowed to begin failing, and after a fixed time period, the refresh cycle is restored. This results in a new bit pattern in memory - the PUF response - based on the random distribution of fast failing cells across the memory block. This approach results in very high-quality responses with strong security properties; however, it suffers from two major drawbacks: it cannot, generally, be performed during system runtime, and it requires a very long time (in the order of minutes) for enough cell failures to produce a good response. Additional researchers went

on to further investigate the DRAM Retention PUF [40, 83].

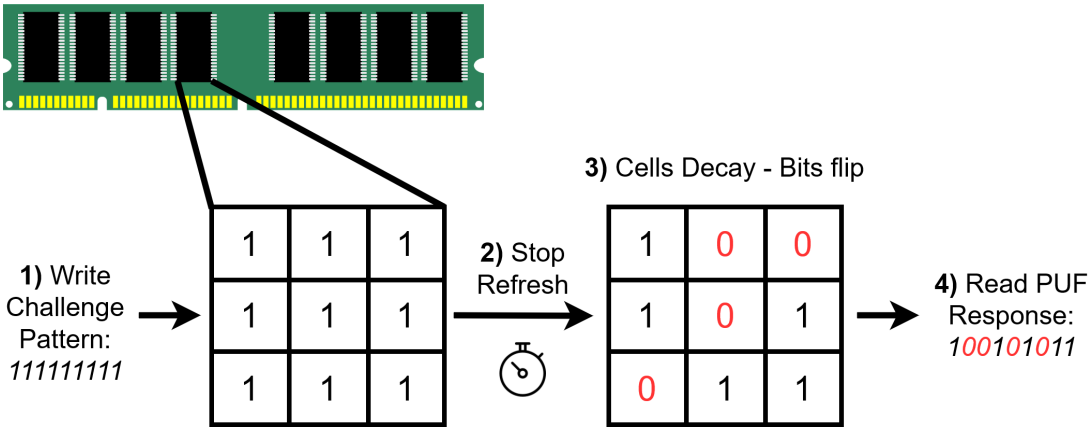

Figure 2.7: *DRAM Retention PUF*

The second key property of DRAM is that being fundamentally a process of charging and discharging capacitors, there is an inherent latency (delay) in read/write DRAM operations. To ensure correct behaviour, there are built-in delay timings for internal operations. These timing parameters can be lowered from their factory defaults, allowing for faster operation at the risk of causing instability. Too significant a reduction places the memory into an undefined behaviour state in which instructions may produce erroneous results. In PUF terms, the important factor is the process-induced variation in the sources of internal latency, such as cell charge and discharge rate, line activation rate, the behaviour of sense amplifiers, etc. PUFs were eventually proposed which exploit this property called DRAM *'Latency'* PUFs, the first of which being by Kim et al. in 2018 [41, 68, 81] and illustrated in Figure 2.8. By reducing or removing delays in the memory controller, internal DRAM operations are given insufficient time to execute, resulting in timing errors. For example, severely reducing the *tRCD* parameter (the required delay between opening a row and being able to access columns in it) and then attempting to perform read operations results in numerous read errors, with the resultant pattern being a product of both the values in the memory being read and process variations in the cell array and sense circuitry. This pattern of errors forms the PUF response, similar to the Retention PUF, and it is likewise the highly random distribution of process variations that makes this a strong identifier for the memory.

In DRAM Latency PUFs, unlike DRAM Retention PUFs, the errors comprising the PUF response are not cell failures. The errors occur from failure to accurately perform operations on the cell contents. If a read operation is used as the trigger, the cell contents remain stable even as the read instructions return erroneous results. Due to this, PUFs of this type can sometimes be used in runtime while maintaining system stability [68]. They also allow response generation in much less time than Retention PUFs. A drawback of this approach is that responses tend towards high noise and are sensitive to environmental conditions, even when compared with other PUF designs [41,68]. This requires more expensive error correction and post-processing to produce a high-quality response. DRAM Latency PUFs were chosen as a key target for the work in the thesis due to the degree of noise in individual responses and sensitivity to some environmental conditions they exhibit. They provide a good example of a PUF with strong potential for practical application but can be limited in performance

by the energy and computational cost of error correction. While the DRAM-PUF is generally designated as 'Weak' (CRP space does not increase exponentially with PUF size), the greatly increased density alongside an supporting more unique challenge patterns increases the CRP space significantly over other Weak PUFs such as SRAM-PUF. We therefore make the distinction of the DRAM-PUF being 'Semi-Weak'.

While many PUF variants have seen various attacks within the literature, DRAM-PUFs have not seen such apparent threats. Being primarily a 'weak' PUF, it is generally assumed that sufficient unique CRPs cannot be extracted from any DRAM-PUF version in order to enable an attack using machine learning (section 2.5), leaving other types of attack open to discussion. Ultimately, the goal of an attacker would be to able to predict outputs of the DRAM-PUF (bit-flip behaviours of cells used for PUF functionality) based on limited knowledge of the challenge data. To achieve this, an attacker requires some method to access some of this memory in order to perform memory reads and writes, which is generally out of scope in a reasonable threat model, assuming that such access indicates root access of an attacker regardless. What requires further experimentation is whether or not there exist correlating factors between bit-cells which flip during PUF operation and other operations/manipulations on the memory. An example of this is rowhammer [42] or data remanence effects (cold-boot attacks) [34]. If, the cells which are vulnerable to these types of attack correlate with the error prone cells during PUF operation, there conceivably exists a predictive attack. This, however, is yet to explored by the research community.

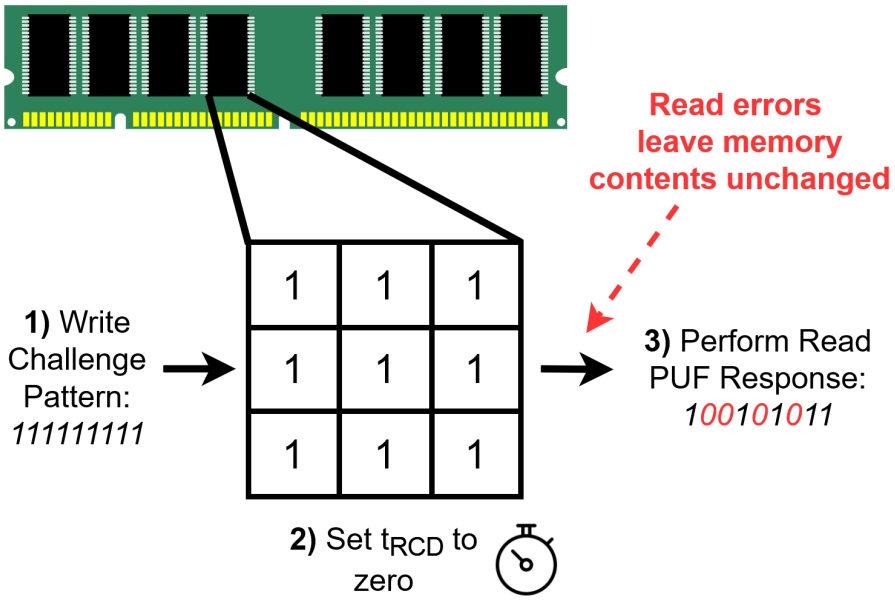

Figure 2.8: *DRAM Latency PUF*

2.5 Machine Learning Modelling Attacks on PUFs

Machine Learning Modelling Attacks (ML-MA) have proven to be a significant security challenge for strong PUFs almost since their first conception [49]. Almost all Strong PUF proposals to date have been demonstrated to be vulnerable to ML-MA [1, 6, 74, 76]. This is

because in a Strong PUF, the large set of CRPs arises from a relatively small number of actual physical variables, i.e. the PUF function is rooted in complex physical factors but is mathematically simple. Even for a large PUF, if some sufficient subset of CRPs is known, it takes relatively little computation to derive a function which produces the same outputs as the actual PUF for any given challenge. The security of the PUF is successfully compromised when an attacker can accurately predict novel responses (outputs) based on new and unseen challenges (inputs) such that they may send the predicted outputs to verifiers, resulting in their successful and, thus, fraudulent authentication. For these reasons, how, or if, Strong PUF designs like the Arbiter-PUF can be reliably secured against ML-MA without introducing excessive complexity remains an open question. While there are many different examples of varying ML algorithms to attack PUFs, we focus on those relevant to the further chapters of this thesis.

2.5.1 Logistic Regression

The first significant attacks on Strong PUF variations was performed by Rürhmair et al. in 2010 on APUFs, XOR-APUFs, Lightweight Secure PUFs (LS-PUFs) and FF-APUFs [74]. In their work, they collected a subset of available CRPs from each PUF type psuedorandomly following a standard normal distribution. This set of CRPs formed the training dataset for various machine-learning techniques in order to model the PUFs, most notably, the Logistic Regression attack.

Logistic regression (LR) is a statistical model used for binary classification tasks which estimates the probability that a given input belongs to a particular category. Unlike linear regression, which predicts continuous outputs, LR predicts categorical outcomes by mapping inputs through a logistic function. The logistic function (also known as the 'sigmoid function´) is used to map predicted values to probabilities out an outcome between 0 and 1, making it ideal for probability estimation, especially for single bit output PUFs such as APUFs. It is defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.6}$$

In LR, the input features $x = (x1, x2, \ldots, xn)$ are combined linearly using weights $w = (w1, w2, \ldots, wn)$ and a bias term b. The linear combination is given by:

$$z = \mathbf{w}^T \mathbf{x} + b \tag{2.7}$$

The probability that the input x belongs to the positive class (first label) is then given by applying the logistic function to z:

$$P(y = 1 \mid \mathbf{x}) = \sigma(z) = \frac{1}{1 + e^{-(\mathbf{w}^T \mathbf{x} + b)}} \tag{2.8}$$

The predicted class label $\hat{y}$ is determined by applying a threshold (commonly 0.5) to the probability:

$$\hat{y} = \begin{cases} 1 & \text{if } P(y = 1 \mid \mathbf{x}) \geq 0.5 \\ 0 & \text{if } P(y = 1 \mid \mathbf{x}) < 0.5 \end{cases} \tag{2.9}$$

Finally, the LR model is trained through minimisation of the loss function 'binary cross-entropy loss´ (known also as log-loss), defined as:

$$\mathcal{L}(\mathbf{w}, b) = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(P(y_i \mid \mathbf{x}_i)) + (1 - y_i)\log(1 - P(y_i \mid \mathbf{x}_i))\right] \tag{2.10}$$

where N is the number of training samples, y_i is the true label, and $P(y_i \mid x_i)$ is the predicted probability.

The model is then iteratively trained using gradient descent in order to minimise the loss function. Each update is given by:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta\frac{\partial\mathcal{L}}{\partial\mathbf{w}} \tag{2.11}$$

$$b \leftarrow b - \eta\frac{\partial\mathcal{L}}{\partial b} \tag{2.12}$$

where η is the rate of learning.

LR for Modelling Arbiter PUFs

Rürhmair et al. applied LR specifically to APUFs. Overall, this task requires prediction of the binary APUF response $R \in 0, 1$ for a given challenge $C \in 0, 1^n$. Each pair of C and corresponding R (CRP) forms an individual sample for training.

Each challenge C is represented as a feature vector which forms $x = (C_0, C_1, ..., C_n)$, where C_k are the bits of the challenge (k being the number of stages of the APUF). Each challenge is assigned the probability $p(C, t|\vec{w})$ that it generates an output (response) $t \in -1, 1$. The vector $\vec{w}$ encodes the internal parameters of the PUF such as the specific runtime delay of the PUF (see section 2.4.1 for details on the additive delay model). The probability of output is then given by the logistic sigmoid acting on the function $f(\vec{w})$. Therefore, f determines through $f = 0$ a decision boundary of equal output response probabilities. Given a set of training CRPs M, the decision boundary is placed by selecting the vector $\vec{w}$ such that the likelihood of observing M is maximal, meaning the negative log-likelihood is minimal:

$$w = \text{argmin}_{\tilde{w}}\, l(M, \tilde{w}) = \text{argmin}_{\tilde{w}} \sum_{(C,t)\in M} -\ln(\sigma(tf(\tilde{w}, C))) \tag{2.13}$$

Finally, $\vec{w}$ is optimised iteratively using gradient information, utilising Resilient Back-propagation (RProp) [9]:

$$\nabla l(M, \tilde{w}) = \sum_{(C,t)\in M} t(\sigma(tf(\tilde{w}, C)) - 1)\nabla f(\tilde{w}, C) \tag{2.14}$$

Using this technique, prediction rates of 99% were achievable on k=64 stage APUFs with as few as 2,555 CRP samples. XOR-APUFs and FF-APUFs could also be modelled within an accuracy of >95%, rising to 99% in all cases where large training set sizes were utilised.

2.5.2 Multi-Layer Perceptron

The Multi-Layer Perceptron (MLP) is a well-studied class of feed-forward artificial neural network [26]. It consists of at least three layers of nodes: an input layer, one or more hidden layers, and an output layer. MLPs are used to solve problems that require supervised learning, where the goal is to learn a mapping from input data to output data based on example input-output pairs. Each layer contains 'neurons', and each neuron in a layer is connected to every neuron in the subsequent layer through weighted edges.

An MLP can be defined from the following components (in order of input to output):

1. **Input Layer:**

   Let X represent the input vector, where $X = [x_1, x_2, ..., x_n]$, with n being the number of input features.

   The input to each node in the first hidden layer is a weighted sum of the input features, denoted as $z_j^{(1)}$, where j is the index of the node in the first hidden layer.

2. **Hidden Layers:**

   Let L represent the total number of layers in the MLP, including the input and output layers.

   For each hidden layer l, let $W^{(l)}$ represent the weight matrix connecting layer l to layer $l + 1$.

   Let $b^{(l)}$ represent the bias vector for layer l.

   The output of each node in hidden layer l before applying the activation function is computed as:

   $$z^{(l)} = W^{(l)} \cdot a^{(l-1)} + b^{(l)} \tag{2.15}$$

   where $a^{(l-1)}$ is the output vector of the previous layer and $z^{(l)}$ is the input vector to the activation function in layer l.

3. **Activation Function:**

   Typically, an activation function σ is applied element-wise to the output of each node in the hidden layers:

   $$a^{(l)} = \sigma(z^{(l)}) \tag{2.16}$$

4. **Output Layer:**

Let y represent the output vector of the MLP.

The output layer applies its own set of weights $W^{(L)}$ and biases $b^{(L)}$ to the output of the last hidden layer:

$$y = W^{(L)} \cdot a^{(L-1)} + b^{(L)}$$

MLP in the Context of a PUF

In the context of a PUF, an MLP is used to model the challenge/response relationship of the PUF, which was first applied in 2017 by Alkatheiri et al to model more complicated FF-APUFs (section 2.4.5) [1]. The PUF's response R to a challenge C can be considered as the input X to the MLP, where the MLP learns to map challenges to responses. The MLP may have multiple hidden layers to capture complex mappings between challenges and responses.

Consider a specific MLP with one hidden layer and a sigmoid activation function. The output of the MLP can be computed as follows:

$$z^{(1)} = W^{(1)} \cdot X + b^{(1)} \tag{2.17}$$

$$a^{(1)} = \sigma(z^{(1)}) \tag{2.18}$$

$$\hat{y} = W^{(2)} \cdot a^{(1)} + b^{(2)} \tag{2.19}$$

Where:

- X is the input vector representing the PUF challenge.
- $W^{(1)}$ and $b^{(1)}$ are the weights and biases of the first hidden layer.
- σ is the sigmoid activation function.
- $W^{(2)}$ and $b^{(2)}$ are the weights and biases of the output layer.
- $\hat{y}$ is the output vector representing the predicted PUF response.

Inb 2017, the authors of [1] were able to achieve prediction rates of up to 96% for FF-APUFs with large numbers of feed-forward loops using relatively few CRPs (max. 200k).

2.5.3 Attacks Exploiting Helper Data

In 2014, Delvaux et al. performed a helper data attack on Pattern Matching Key Generators, exploiting their helper data [17]. They demonstrate a significant risk to full key recovery in an experiment on 4-XOR Arbiter PUFs. More recently, helper data attacks utilising ML have been proposed to exploit the publicly known helper data used for error correction to predict PUF responses. Streider et al. demonstrated in 2021 that public challenge data and helper data alone is sufficient to mount a successful modelling attack on a PUF, even with as few as 800 helper data bits [79].

2.6 PUF Obfuscation

In response to the developing threats posed by ML-MA, schemes were proposed that aimed to *'obfuscate'* the linear relationship between the PUF challenges and responses to weaken the effectiveness of ML models to learn relationships between them. By including additional logic surrounding the input and output interfaces of a vulnerable PUF design, it is possible to reduce the potency of ML-MA, where attackers can access initial obfuscated PUF inputs and final outputs. Figure 2.9 demonstrates the assumed threat scenario when designing PUF obfuscation schemes. In this case, attackers are given free access to the initial challenge $C^{'}$ and can measure the final response $R^{'}$. The interface to the PUF underlying the design is assumed to be inaccessible, and the PUF could be modelled using C and R. The security game is satisfied if, given a non-linear relation between $C^{'}$ and $R^{'}$, the adversary gains a negligible advantage in modelling both the underlying PUF and the obfuscated PUF, such that novel responses cannot be predicted for new challenges.

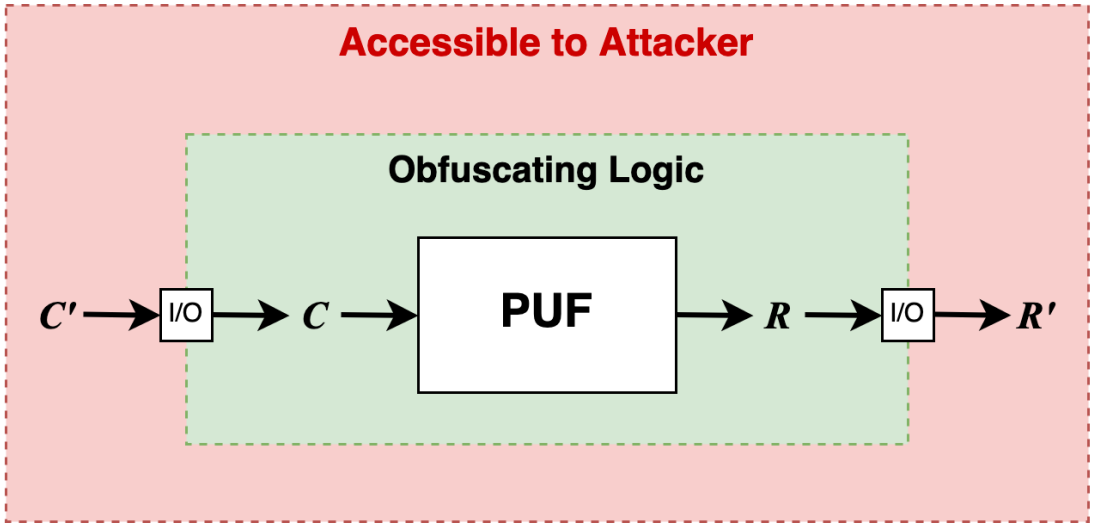

Figure 2.9: *Example of PUF obfuscation concept*

The first type of obfuscated PUF was the Controlled PUF, designed by Gassend et al. in [28] in 2008. This design utilised hash functions and xor logic to control the output of the PUF. While capable of diminishing the effectiveness of ML-MA, this design has a high hardware resource cost, which is a significant issue when considering deployment on particularly resource constrained systems. Additional schemes have since been proposed using various methods in hardware, which will be discussed in more detail in Chapter 4.

2.7 PUF-based Authentication

While PUFs have been proposed to be suitable for many security applications, one of the most significant and researched is for authentication purposes. As PUFs can be used to prove the presence of a device, they are useful when measuring their unique outputs to prove the authenticity of a device, allowing the creation of secure sessions at a relatively low cost for embedded security applications.

Authentication protocols that utilise PUF are deployed in two phases: an enrollment phase and an authentication phase. The enrollment phase, demonstrated in Figure 2.10, occurs within a secure server environment where it is assumed that device-to-server communication remains secure for the duration. Here, a large set of CRPs is generated by monitoring the responses for given challenges to a device's PUF and storing them in a CRP database on the server. Once a sufficient dataset has been collected, the devices can be deployed into their 'real world' roles. From now on, when a device wishes to communicate with the secure server, the Authentication phase begins. Practically, all PUF responses to be used for authentication must be captured in the initial enrolment phase. Once all enrolled CRPs are expended, the PUF device must either be discarded or recalled for retrofitting with a new PUF (or, measurement of new unique CRPs). It is feasible to design a protocol whereby once an authenticated session is established with the device, a new set of CRPs can be transmitted to the server without the need to recall the device, however, this assumption can only be made if the protocol is supported by a form of symmetric encryption.

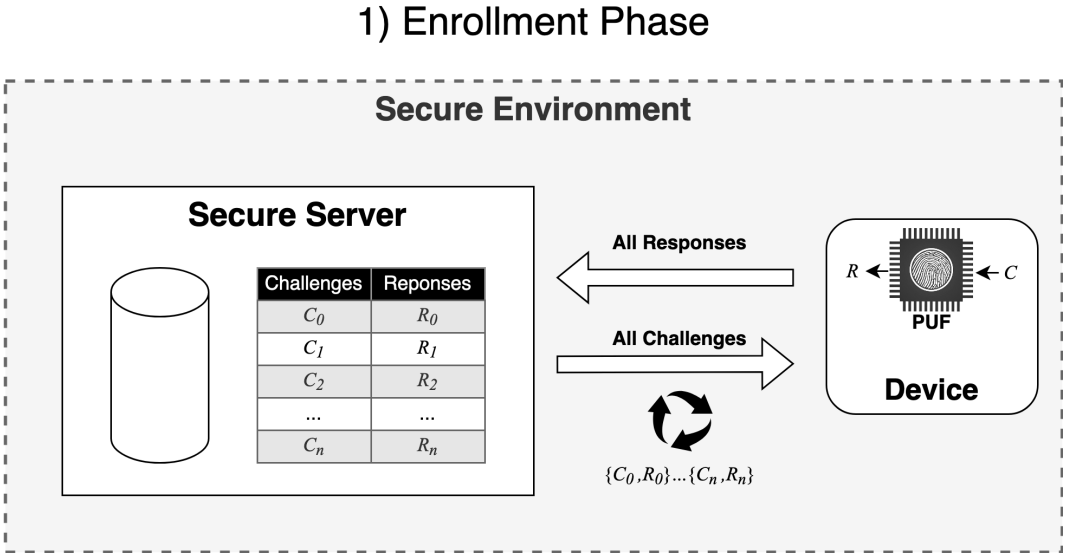

Figure 2.10: *CRP-based PUF Enrollment Phase*

Within this phase, shown in Figure 2.11, the device communicates openly with the secure server. The secure server selects a challenge from its database for the device wishing to communicate and sends it to the device. Once the device receives the challenge from the server, it issues it to its PUF, records the response, and sends it to the server. The server finally checks the response sent by the PUF to the response stored in the dataset. If they match, authentication is passed, and a secure channel is opened. If they do not match, the authentication attempt is rejected.

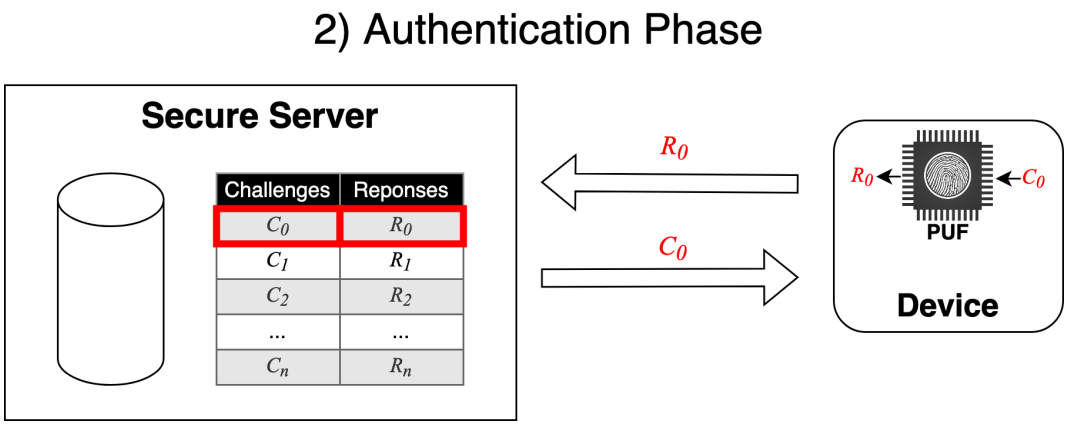

Figure 2.11: *CRP-based PUF Authentication Phase*

2.8 Cryptographic Functions

We define particular cryptographic functions which are relevant for the technical chapters of this thesis.

2.8.1 One-Way Function (OWF)

A OWF is an important cryptographic primitive and has been widely used in modern asymmetric cryptography, which is easy to compute but difficult to invert. In mathematical terms, a $\mathsf{OWF}(\cdot)$ is a function from a set of data objects to a set of values having the following properties [87]:

- Given any value v, it is computationally infeasible to find a data object d such that $\mathsf{OWF}(d) = v$.
- Given any data object d, it is computationally infeasible to find a different data object d' such that $\mathsf{OWF}(d') = \mathsf{OWF}(d)$.

2.8.2 Key Derivation Functions (KDF)

KDFs are essential primitives in cryptographic systems which transfer a source of randomness into cryptographically strong secret keys, and sometimes the initial materials do not have to be distributed uniformly. According to the extract-then-expand approach, a KDF has two components: *extractors*, which extract a fixed-length key K from entropy sources, and *random functions*, which expand K into several additional pseudorandom cryptographic keys. KDFs take four inputs: a random seed r, a length L, salt s and context c, and return an L-bit key k. The salt value is essential to obtain generic extractors and KDFs that can extract randomness from arbitrary sources with sufficiently high entropy in the case where inputs are non-uniform. Psuedo-random functions (PRFs) are similar to KDFs and can be used as such. They are often less computationally complex; however, *require* highly uniform secret keys to guarantee security.

Chapter 3

PUF-Phenotype: A Robust and Noise Resilient Approach to Aid Group-Based Authentication With DRAM-PUFs Using Machine Learning

As the demand for highly secure and dependable lightweight systems increases in the modern world, Physically Unclonable Functions (PUFs) continue to promise a lightweight alternative to high-cost encryption techniques and secure key storage. While the security features promised by PUFs are highly attractive for secure system designers, ML-MA continues to pose a significant threat against PUF authentication schemes. More recent ML-MA have even exploited publicly known helper data required for PUF error correction in order to predict PUF responses without requiring knowledge of response data. In response, research is beginning to emerge regarding the authentication of PUF devices with the assistance of ML as opposed to traditional PUF storage techniques and comparison of pre-known Challenge-Response pairs (CRPs). In this Chapter, we introduce a classification system using ML-based computer vision techniques based on a novel *'PUF-Phenotype'* concept to accurately identify the origin and determine the validity of noisy memory-derived (DRAM) PUF responses as an alternative to helper data-reliant denoising techniques. As opposed to previous work in this field, we perform classification over multiple devices per model to enable a group-based PUF authentication scheme. We achieve high classification accuracy using a modified VGG16 deep convolutional neural network (CNN) for feature extraction in conjunction with several well-established classifiers. Our results demonstrate that our system can effectively distinguish the authenticity of even highly noisy DRAM-PUF responses for authentication purposes.

3.1 Introduction

IoT devices are an unavoidable and growing presence in modern life and have been adapted for various applications, ranging from domestic use to healthcare, autonomous vehicles, and even

military systems [46]. Many of these applications have high security and privacy requirements due to the sensitivity of the information they transmit or store. Providing adequate security to such systems is more challenging due to heavy restrictions in the availability of computational, storage and network resources, making traditional security methods often inaccessible due to their high complexity. As a result, there is a requirement for robust, lightweight solutions to address security concerns. PUF identities can be used for memory-less secure key generation or directly to generate single-use authentication tokens for device verification in a challenge-response protocol. This type of authentication can be particularly useful for IoT scenarios, such as home IoT, where devices are often communicating directly with one another (device-to-device). As PUF entropy is extracted from small physical variations in the circuitry, environmental effects such as temperature, voltage variation, and temporal effects (ageing) can introduce noise in the PUF response. To overcome this, a degree of error correction is required in most PUF designs. A common approach is to use Helper Data Algorithms (HDAs), which map PUF responses to code words of an Error Correction Code (ECC) scheme. Helper Data (HD) is required during this process, which must be assumed to be publicly known. Using the HD alongside a given HDA enables useful data to be extracted from the noisy PUF data, such that an expected value is retrieved.

3.1.1 Related Work

While error correction methods are effective for reducing noise from measured PUF responses, in some implementations, this can lead to the introduction of new vulnerabilities which target the error correction mechanisms themselves. Recently, a new problem has emerged from this issue, namely *privacy leakage through HD* [79]. In a standard PUF threat model, challenges and HD are assumed to be public and, therefore, accessible to an attacker. An HD attack is mounted by an attacker who only requires access to challenge data and HD and allows them to predict response data and thus compromise the PUF secret. This is in contrast to more well-studied modelling attacks, where knowledge of the input challenge and matching PUF response is required for some large number of challenges to comprise other unseen response secrets [16, 74]. PUF authentication schemes are often accompanied by an extensive security analysis against a common PUF adversary model (which typically includes modelling and/or side-channel attacks); however, these schemes often also integrate a requirement for HDA and/or ECC to deal with noisy PUF responses [31, 33, 47]. An issue with such systems is that they cannot sufficiently provide a defence against this new type of HD attacker, who only requires publicly available data to model the PUF successfully. It is, therefore, essential for techniques to be developed that can counter such an attacker while simultaneously ensuring security and resource requirements are met. While it is common practice for PUF security systems to employ HDA and ECC, it is not a strict necessity but rather an established industry-accepted approach for denoising PUF responses.

To achieve sufficient noise reduction by different means, it is intuitive to look to fields of study where noise reduction is a key focus, such as computer vision. In computer vision, denoising and classifying noisy data are a well-studied topic. Machine Learning (ML) is a powerful tool used both for extracting noise from images, such as feature extraction through Deep Denoising Autoencoders [50, 51] and image classification of already noisy images [73]. The problems solved through these techniques in computer vision are almost directly represented by the noisy PUF problem, namely the requirement to either actively remove noise

from PUF measurements or classify the PUF measurement regardless of the noise. One type of PUF where this is particularly relevant is DRAM-based PUFs, which produce very large amounts of PUF data but with a high degree of noise. With large response sizes, it is intuitive to represent responses as images to test the effectiveness of computer vision techniques to solve such problems.

In [70], Deep Convolutional Neural Networks (CNNs) are utilised to classify noisy DRAM-PUF responses using image data of the PUF measurements. Their solution uses a custom memory controller to precisely measure PUF responses and map the physical bit-cell layout directly to a grayscale image, which *directly represents* the real DRAM properties and structure. In this approach, a very large dataset of DRAM-PUF responses are used in training, as input patterns are based on many arbitrary bit string inputs to the memory. In practice, such responses are unsuitable for security purposes, as with the DRAM-PUF, only three types of unique input value exhibit suitable entropy on anlysis of the final response (discussed in detail in Section 3.2.2). Yue et al. proposed a similar approach, using very large raw responses from DRAM start-up values to authenticate PUFs without an explicitly stored CRP database [94]. In this approach, various CNNs are tested to classify three DRAM PUFs and extract features to be compared with features known by a trusted entity.

These works have focused on the learnability of raw memory-based PUF data, where CNNs extract features from image data that directly represent the physical organisation of the bit-cell matrices on the chip. However, achieving accurate image representations of measuring responses from DRAM is a difficult task, especially during the run-time of a device. In addition, potentially useful information could be stored unknowingly in such images - for example, charge leakage paths for capacitive cells - which could in theory, provide an attacker with information with which to predict cell-failure behaviour of the PUF.

It is important to note that one overlooked advantage of ML-based authentication is the removal of CRP database storage requirement, which has some useful implications. As an attacker can read CRPs, it must be assumed in traditional PUF protocols, the CRP database is stored on a system in a trusted environment (Section 2.7), limiting device-to-device authentication as devices must be assumed to be publicly accessible (and therefore vulnerable to attackers accessing the CRP database). By offloading the task of response scrutiny to an ML model, storing responses on the prover system for comparison with received responses is no longer required. This can expand current limitations dictated by non-ML PUF-based authentication schemes. This property also provides a potentially significant advantage over many current schemes, where key data must be stored on devices.

3.1.2 Motivation

While the previously mentioned schemes are proposed as an alternative to error correction algorithms, this is not without trade-offs. While the HDA and HD computation and storage requirement is removed, a new computation and storage requirement is introduced through the requirement to train and distribute an ML model. The resource-intensive training of the models is performed just once during enrolment; however, on-device verification still requires potentially costly model storage and execution. If a device is required to have relatively high computational performance to be compatible with such schemes, the question stands as to whether a PUF-based solution has any advantage over traditional handshaking protocols. Furthermore, the previously mentioned schemes do not fully exploit the benefit of the removed

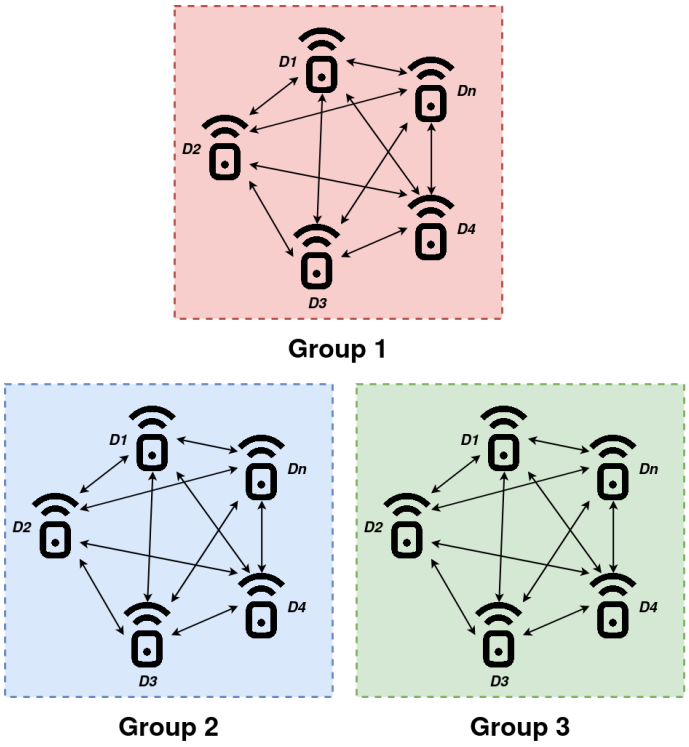

Figure 3.1: *Intra-Group Communication Environment*

CRP database requirement.

Each scheme proposes single-device authentication with a verifier, which must be in a trusted environment. These aspects reduce the application of such systems to more traditional device-to-server authentication as opposed to device-to-device. This restriction limits the PUF-based authentication from wider IoT scenarios, for example, where a group of devices require inter-node communications (such as Home IoT) within the group without explicit contact with an entity in a trusted environment (Figure 3.1). Finally, current ML-based PUF authentication only considers a single device per trained model; therefore, considering a group of n provers, a given verifier is required to store $n - 1$ models in order to handle authentication requests from each other group member. This is an unrealistic requirement for on-device authentication, given the resource constraints of an IoT device.

Each of these points raises the question: is it possible to apply a *single* ML-based PUF authentication model that is appropriate for *multiple* grouped devices, that can run on lightweight devices and without a third-party trusted verifier?

3.1.3 Contribution

In this Chapter, we present a method for authenticating a group of PUF-enabled devices using a modified CNN and standard ML classifier in combination. This method is more lightweight than existing comparable schemes. The scheme operates on images formed directly from noisy PUF responses without helper data, knowledge of the physical properties of the PUF, or its design. This allows for the use of a single classifier with low storage requirements, which

can run on relatively low resource devices, and which can natively perform both group-based authentication (*is device X part of group Y?*) and device-specific authentication (*is device X itself authentic?*).

The proposed classification architecture also avoids using data physically representative of the internal PUF structure but rather an image derived from a non-design-specific response data stream. As it relies only on externally observable characteristics and not knowledge of the underlying structures, treating the PUF as a black box, we call this approach a *PUF Phenotype*. Section 3.2.1 will introduce this concept in more depth. The major contributions of this chapter are as follows:

- The concept of *PUF Phenotype*, a *new* classification approach, where PUF identity is the full externally observable PUF behaviour, including noise, without supplementary knowledge of PUF structure, physical properties, or environment.
- DRAM Phenotype-based Authentication Network (DPAN), an ML-based group classification model, uses model confidence to accurately identify and authenticate devices without needing Helper Data Algorithms, Error Correcting Code or pre-filtering algorithms. To the best of our knowledge, this work is the *first* to consider a single ML model for multi-device PUF authentication.
- Verification and analysis of the proposed scheme on *our own* experimental dataset, using noisy DRAM-PUF responses collected under a broad range of environmental conditions. We provide this dataset as an open resource for the research community.
- Evaluation of classifier accuracy of DPAN for varying classifiers and group sizes up to 5 devices, demonstrating that good performance can be achieved without needing device-specific classifiers.
- Performance analysis of a DPAN implementation on a resource-constrained system with power consumption analysis, storage requirements, and execution time.

3.1.4 Chapter Organisation

In Section 3.2 we introduce our proposed scheme, with our novel PUF Phenotype concept, dataset and classification methodology. Section 3.3 provides the results and a discussion of our experiments using various classifiers and numbers of devices. We also provide a performance analysis of our model executed on a lightweight Raspberry Pi system in section 3.3.4. Finally, Section 3.5 concludes the work and identifies areas for further research.

3.2 Proposed Scheme

This section presents our proposed scheme, which consists of three key parts: The PUF Phenotype concept, the group-based authentication setting for our work, and our DRAM Phenotype Authentication Network: DPAN.

3.2.1 PUF Phenotype

Biometric-based identity is a field adjacent to and highly relevant to electronic PUFs. In a biological context, there are two sets of connected identifying characteristics: Genotype and Phenotype. The genotype consists of a person's actual set of genetic instructions. At the same time, their phenotype is the set of externally observable traits which arise from the interaction between their genotype and the environment [82]. A specific example of genotype vs. phenotype would be as follows:

Genotype: The unique sequence of DNA *instructions* that exist to determine face shape.

Phenotype: The resulting detectable expression (appearance) of the face, effected by environment.

While the phenotype is primarily derived from the genotype, the phenotype is *not synonymous* with the genotype, as the genotype cannot influence the effect the environment has on the emergent properties of the phenotype. In biometric authentication, knowledge of the genotype is not required. If we consider the case of facial recognition, the computer vision system only has access to the *externally observable characteristics* - the phenotype, plus environmental noise (incident light, angle, etc.). This distinction is described in Figure 3.2. Despite this, such systems can perform authentication with high accuracy. Considering that a biometric is a form of Extrinsic PUF, this raises the question of whether, in the area of electronic PUFs, the costly on-device error correction and helper data (which may leak information about the PUF) which we often rely on is universally necessary for accurate authentication?

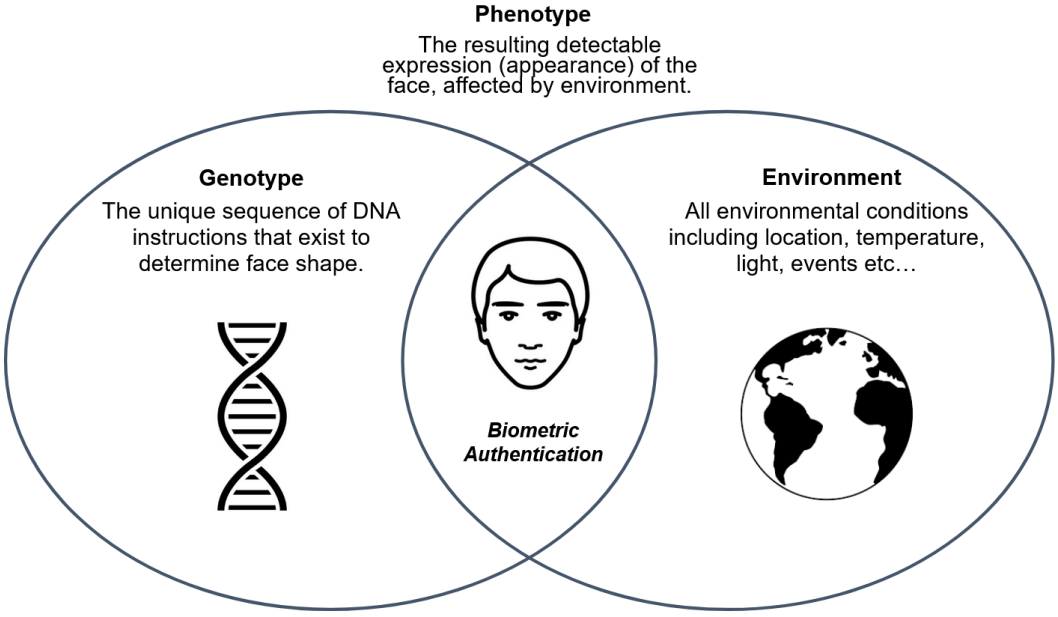

Figure 3.2: *Phenotype/Genotype distinction for human biometric authentication*

Shown in Figure 3.3, drawing inspiration from the field of biometrics, we propose that an analogous distinction for electronic PUFs should be considered as an aspect of protocol design. That is, whether to consider the underlying structure of the PUF and try to remove noise so that only this underlying structure contributes to the identity or whether to use a PUF 'Phenotype' where both the PUF behaviour and the PUF response to environmental changes (i.e. noise) are treated as one set of identifying characteristics. The former is the default assumption in most PUF research, but both approaches have advantages and disadvantages. This is not to say that error correction-based approaches should be discarded, but rather that more significant consideration should perhaps be given as to whether this default approach is always optimal.

We express the idea of phenotype concerning PUF in the generated images of our DRAM-PUF measurements. The physical organisation of the bit-cell matrix on the DRAM module and the transistor level behaviour is analogous to the genotype, while the PUF response image generated from that structure, including noise, is the expressed phenotype. The phenotype is not a visual representation of the PUF's physical structure (as was used in [70, 94], for example) but rather a structure-agnostic visualisation of the PUF identity. So long as the same method of image formation is used consistently, the phenotype can be generated from PUF data from any source, meaning knowledge of the PUF structure or even the type of PUF in use is irrelevant. Multiple PUF designs can be used in one system, with a unified enrolment procedure, authentication database, and verifier. In addition, the phenotype includes environmental and inherent noise, reducing or negating the need for on-device error correction or the use of helper data. It is demonstrated in Section 3.3 that using this approach, devices can be authenticated with a high degree of confidence, even under highly variable environmental conditions and without any error correction mechanism.

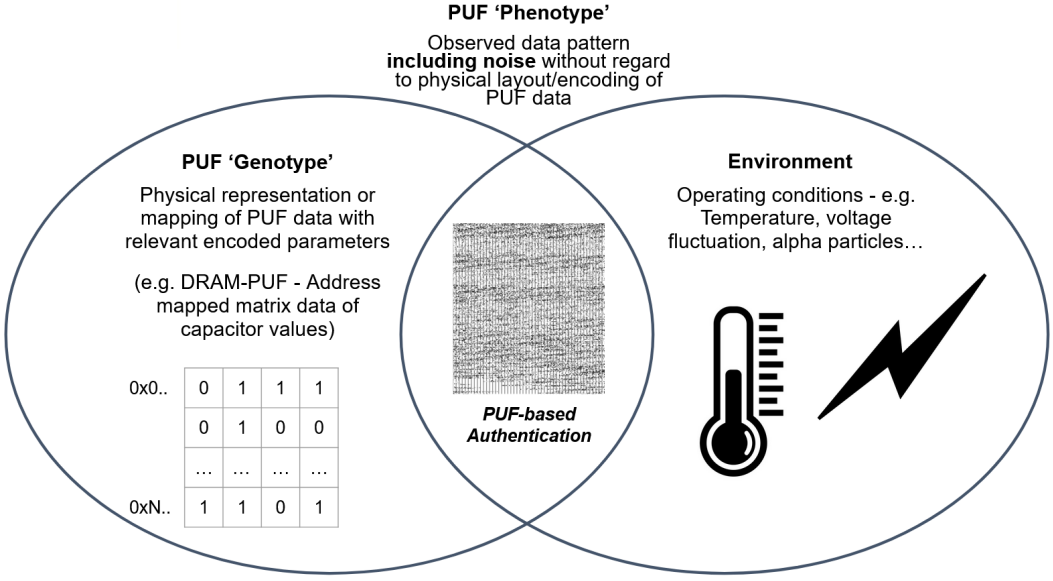

Figure 3.3: *Phenotype/Genotype distinction for PUF-based authentication*

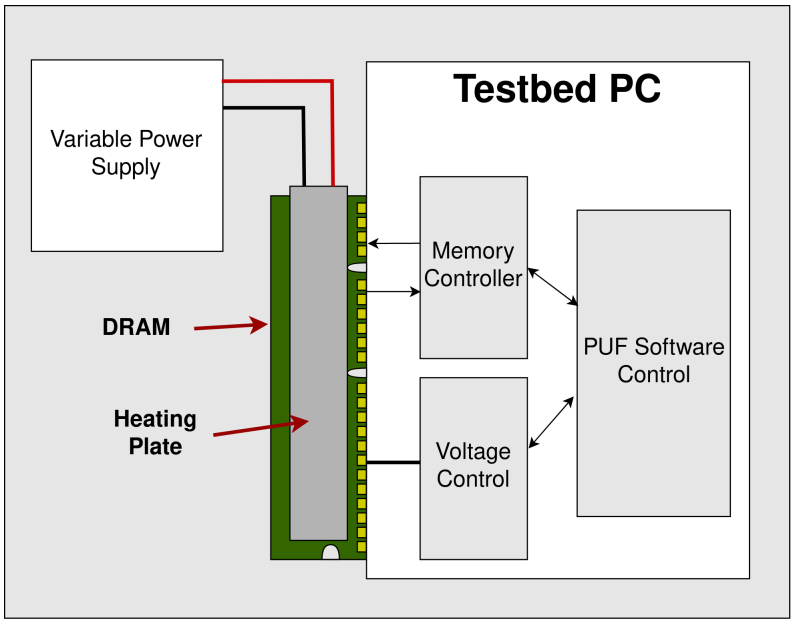

Figure 3.4: *Experimental Setup for Dataset Generation*

3.2.2 Phenotype Dataset

For the demonstration of this work, we collected our own dataset of DRAM Latency PUF responses gathered from a test bed based on a Commodity Off-The-Shelf (COTS) AMD A series processor, controlling COTS DDR3 DRAM modules in DIMM form factor, using only software methods to acquire the PUF response*. Specifically, the Latency PUF method used was lowering the t_{RCD} timing to 0 and attempting sequential reads of the target memory, using the method described in [68] (functionally described in Section 2.4.7). The test bed is effectively a small form factor desktop computer and runs a live version of Ubuntu Linux during experimentation. It should, therefore, closely reflect the performance and behaviour of the DRAM Latency PUF in a real-world scenario. Four QUMOX-branded DDR3 DIMMs were tested, each comprising eight individual DRAM chips. For PUFs of this type, differences in the PUF response between manufacturers have previously been noted [68]; when using appropriate post-processing or mixed input patterns, the difference is relatively minor, meaning this dataset should be reasonably representative of DRAM Latency PUFs in general. The chips under test were characterised by targeting a representative physically contiguous section of 4Kb from each chip's starting point. The DRAM Latency PUF is data-dependent, so three input patterns were used in each experiment: 0xFF, 0x00, and 0x55.

DRAM-PUF Data Dependence Writing an 0xFF, 0x00 and 0x55 input pattern to the DRAM translates to an area of DRAM memory which (before lowering the t_{RCD} timing

*The dataset used in this work is provided as an open resource for the research community, accessible at [60]: `https://doi.org/10.15131/shef.data.26977528.v1`. Additionally, all dataset manipulation code can be found at: `https://github.com/owenmillwood/DRAM_PUF_Analysis`. Finally, source codes for experiments carried out can be found at [61]: `https://doi.org/10.15131/shef.data.27094222.v1`.

parameter) contains all ones, zeroes and a mixed zero and one checkerboard pattern, respectively. This provides three distinct challenge patterns that can be used per physical DRAM location for PUF operation. While it would intuitively appear highly useful to apply random challenge patterns to the memory to increase the unique CRP space, it is important to note that such patterns would not be suitable to generate PUF responses due to the various entropy sources of the DRAM-PUF and, consequently, the influence each source has on overall entropy. There are three sources to consider (in order of most to least influence on entropy):

- The value stored in a cell, i.e. whether the cell is charging from the bit line (0 held in cell) or draining into it (1 held in cell).
- The values in surrounding cells, due to charge leakage; Whether they are leaking charge out or drawing a small amount in influences the value of a given cell. The 0x55 checkerboard pattern is distinct from 0xFF and 0x00 because it alternates these influences instead of all of one type.
- External influences such as gate transistor leakage, line variation, sense amplifier variation, and variation in the memory controller are difficult to distinguish from environmental noise.

The influence that the surrounding cells have on the entropy (likelihood to flip) of a given cell is far smaller than the original value in the cell itself for most cells, meaning randomly sequenced challenge patterns do not behave dissimilarly to the 0xFF, 0x00 and 0x55 patterns [68]. Given an attacker obtaining knowledge of the cell's 0xFF, 0x00 and 0x55 behaviour, correctly predicting most of the output to a given complex input pattern would be a trivial task. The observable high-level patterns (features) that emerge from the physical variations is a probability of failure for each cell in the memory, influenced by the input pattern due to the complex interaction of leakages in neighbouring cells. Cells with a probability of failure less than or greater than 50% are reliable, while cells close to 50% probability of failure are noisy and unreliable. These factors have been captured in the generated dataset, which emerge as features, meaning computer vision models do not require underlying knowledge of the physical PUF features (charge decay rate, etc.) in order to perform classification. For this reason, it is reasonable to apply the technique demonstrated in this work on other types of PUF with similar size responses.

3.2.3 Image Processing

Each of these chip measurements were preprocessed into grayscale images to produce the dataset for training our model. Initially, we convert each pair of two hexadecimal digits (four pairs per DWORD line) into an integer ranging between 0-255. This integer denotes the intensity value (black to white) for a single pixel in the image, which is used to represent the response data visually. The size of each response measurement provides enough data to produce a grayscale image of size 220x200 pixels. Each image was labelled with a class name corresponding to the device it was generated from. Rather than a numerical label, we chose the arbitrary labels Alpha, Beta, Delta, Gamma and Epsilon for readability. These final DRAM-PUF response images are what we refer to as *PUF Phenotype*. We name this process *IMGEN*, described in part B of Figure 3.5.

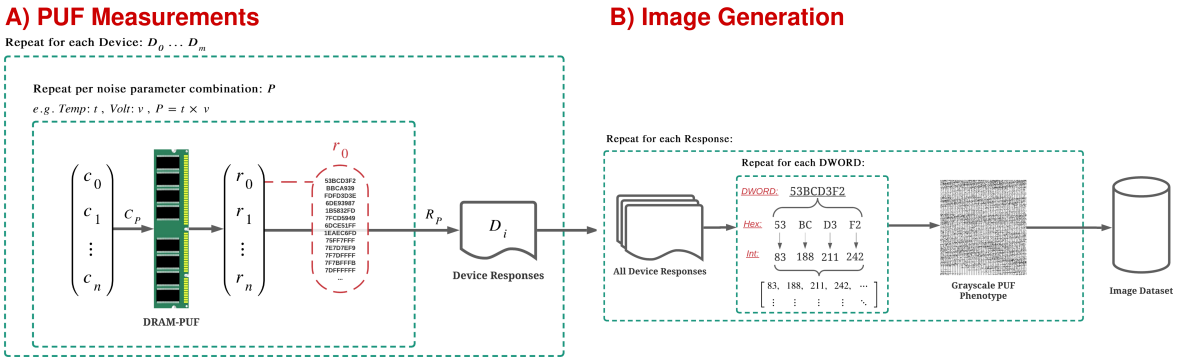

Figure 3.5: *DRAM-PUF Phenotype Data Generation*

3.2.4 Measuring Noise

To measure the impact of environmentally-induced noise, the response for each input pattern was characterised at increasing temperatures, starting at a baseline of 20°C and increasing in 10°C intervals up to a maximum of 50°C (20°C, 30°C, 40°C, 50°C). This process was repeated at a nominal DRAM voltage of 1.5v and a reduced 1.27v. The corner case of reduced voltage and high temperature produces the most noisy responses expected from a PUF of this type and configuration in practice. This provides a robust test for a noise-tolerant authentication system such as the one presented in this work. It should be noted that highly reduced temperatures (0°C and below) would also introduce increasing instability to the DRAM [84]. Due to equipment constraints, generating data at lowered temperatures is out of scope, however, the noise generated via lowered temperatures would manifest similarly across the responses as when higher temperatures are used (randomly distributed noise). The experimental setup for collecting our dataset is shown in Figure 3.4. The original measurements are formatted as a list of 32-bit DWORDs in hex format (eight characters, e.g. FFFA3F6C) as read from memory itself. This first process is described in part A of Figure 3.5 and forms the scheme's Enrollment phase as described in Algorithm 1.

Figure 3.6 compares the hamming distance (noise) between a baseline measurement, a second measurement at ideal conditions (20°C, normal voltage), and a third measurement at worst case conditions (50°C, low voltage)[†] As these measurements were taken with the same challenge, the images would be identical if the noise were not a factor. On comparison, the repeated measurement in ideal conditions showed a similarity of 94.05%, demonstrating an intrinsic noise of around 5.95%. For the worst case, the noise effect is much more significant, with a similarity of 63.09%, i.e. 36.91% noise, an increase of >40% from ideal conditions. If we similarly compare the ideal response of each chip in the dataset with measurements over the full temperature range (ideal +10-30°C) and for both high and low voltage, we can observe actual noise values in the range 6.5-93.5% and with a mean value of 24.8%. In the subsequent sections, we will demonstrate that, even with this high degree of noise, a properly trained model can still recognise the unique features of each PUF in a set to distinguish them both from each other and PUFs outside the set or faked responses.

[†]**Both temperature and voltage effect noise in DRAM-PUF measurements. Measuring a response at a high temperature and lowered voltage represents the most extreme case of a noise-inducing environment achievable by our dataset.**

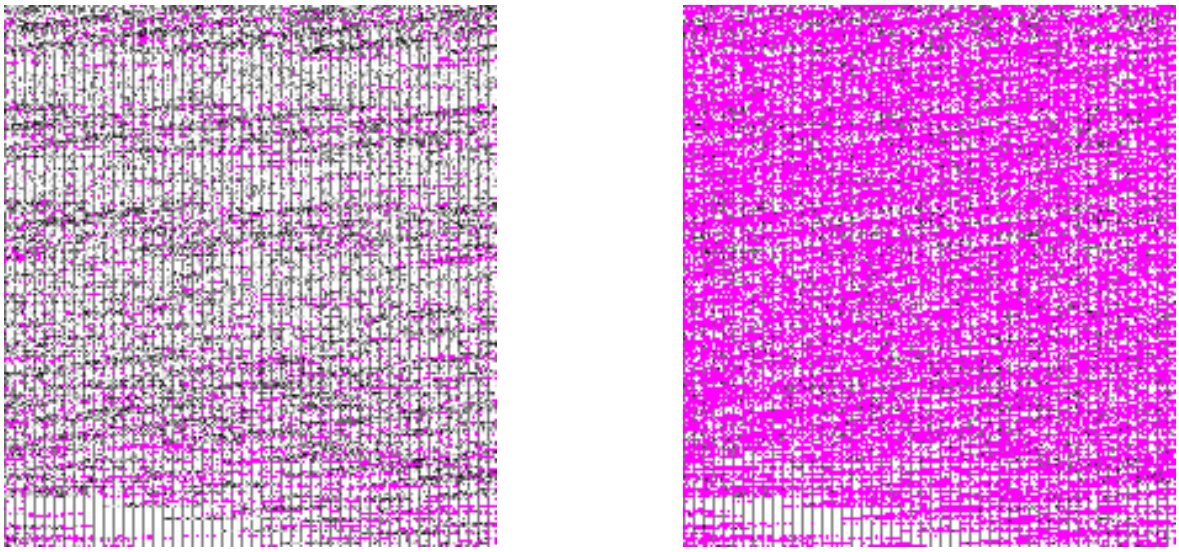

Figure 3.6: *Side-by-side Comparison of DRAM-PUF Measurements (noise highlighted in pink)* ***Left:*** *Response Measured in Ideal Environmental Conditions (5.95% Noise)* ***Right:*** *Response Measured in Extreme Environmental Conditions (63.09% Noise)*

Table 3.1: *Maximum, minimum and average hamming distance (HD) between baseline and repeated measurements (noise) across the entire latency PUF dataset.*

Mean HD	Minimum HD	Maximum HD
0.248	0.065	0.935

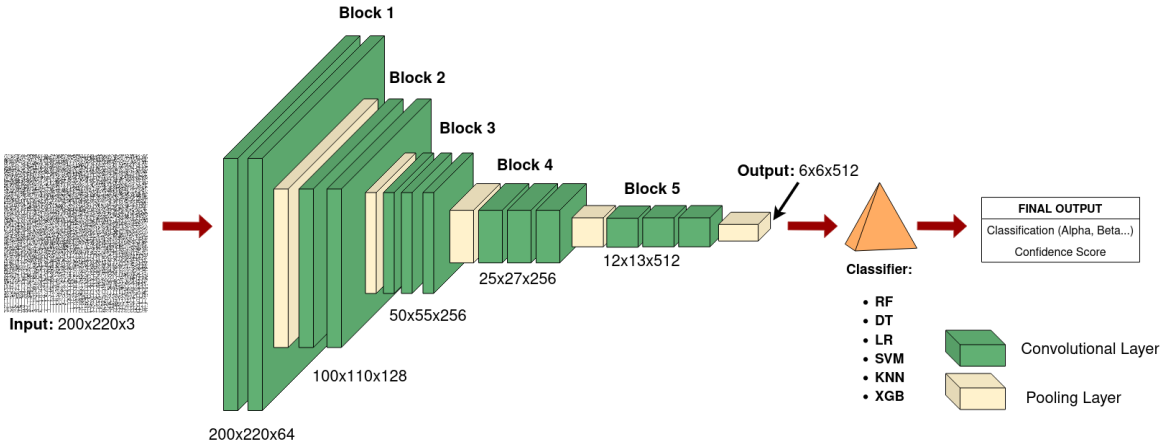

Figure 3.7: *DPAN Architecture*

Algorithm 1: DPAN Enrollment Phases

Dataset Generation

Input: $C \in \{c_0, c_1, ..., c_n\}$: *Challenge Data*
Output: X: *Phenotype Dataset*
Data: P: *Environmental parameters* ($Temp$, *Voltage*)
m: *Number of devices*
n: *Number of challenges*

```
for i ← 0 to P do
    for j ← 0 to m do
        for k ← 0 to n do
            Write Challenge Pattern from Start Location
            Set t_RCD to 0
            R_ijk ← Perform Read operation on DRAM block
            x ← IMGEN(R_ijk)
            /* x:  PUF Phenotype                                   */
            Assign device label to T
            Store x in X
Return X
```

Model Training

Input: X: *PUF Phenotype Dataset*
Output: $DPAN$: *Trained model*
$\tilde{t}$: *Confidence threshold*

```
Split X into train/test sets: o & p
F_a ← Fine-tune VGG16 using o
/* F_a:  Training features                                         */
F_b ← Fine tune VGG16 using p
/* F_b:  Testing features                                          */
Train Classifier using F_a
Test Classifier using F_b
DPAN ← Combined trained VGG16 & Classifier
t̃ ← Tune confidence threshold to zero false positives return DPAN, t̃
```

3.2.5 Intra-Group-Based Authentication Setting

PUF-based authentication typically operates on a per-device basis. However, in some use cases, it is desirable to confirm device identity and determine whether that device is a group member, for example, whether a given device is part of a group with elevated access privileges. This can be done in existing PUF schemes layered atop individual device authentication. This works well in scenarios with a highly resourced central verifier because it can be assumed the verifier can safely handle large amounts of data which must be kept secure from adversaries (e.g. CRP databases, group membership lists) and has substantial computational resources. However, when authenticating directly between IoT devices, this poses significant challenges. In such a scenario, storage space and computational resources are minimal. Further, they

operate in a less secure environment, which limits what data can be safely stored on the device, given that an insecure device could provide an entry point to a secure server for an attacker. Based on these observations, we determined that a computer vision-based ML classification scheme operating on the PUF Phenotype data described above can train a model to perform both individual device and group authentication. When presented with a PUF response, the model can determine first whether it is a group member and if so, which group member in particular without any additional computational or storage requirements and furthermore, that this model is small and lightweight enough to allow direct device-to-device (intra-group) authentication in this manner. Using this, a single model may enable the identification of multiple devices in a single model, saving the device from requiring an individual trained model to be stored for authentication of each other individual device within its group.

To test this concept, we assume the following scenario:

- There is a network of low-resource IoT devices deployed in groups of up to 5 devices. The limitations of this method in regards to group size are discussed in section 3.2.7.3
- Each device has a PUF, which has been characterised before deployment as described in section 3.2.1.
- For each group, a model has been trained prior to deployment using this data. This training is described in section 3.2.7.
- The model for each group is stored on each device in the network. This feasibility in terms of storage requirements is analysed in Section 3.3.4.
- Devices perform both individual and group authentication directly with no central verifier system. The computational and power costs of this are examined in Section 3.3.4.

3.2.6 Confidence Decision Threshold

When considering solutions for authentication, false positives and false negatives have different implications for the system. False negatives, where a legitimate entity is incorrectly denied authentication, impact system performance as a new handshake must be attempted. False positives, however, where a malicious user is granted authentication, are far more important to consider. As an attacker would be free to query our system, nothing prevents them from sending a fabricated image to our model to force it to classify it as one of the legitimate classes. In order to combat this type of attack, we utilise the model confidence to filter to authentication attempts. By applying a confidence threshold, received Phenotype images input to the model must reach a minimum level of confidence that the model must achieve to allow authentication. A threshold may be set to enable authentication if a given classifier demonstrates high confidence on average and low confidence on incorrect classifications. This is to say, when a model outputs low confidence, it is indicated that the model is unsure whether the classification was correct. This provides two possible situations: the received image is from a legitimate device and the received image is fraudulent. In the first case, a low confidence score at worst causes a false negative (a legitimate device is denied authentication). In the infrequent event that a legitimate device is rejected, they may initiate authentication again and succeed with a high probability. In the second case, an adversary

would ideally be unable to produce their own responses, allowing for low model confidence on classification and enabling a higher true negative rate (fraudulent attempts correctly denied authentication). Considering this, a suitable *confidence threshold* can be obtained from the average confidence score for both incorrect classifications of legitimate images and the arbitrary classification of fraudulent images. Such a threshold value would have to be obtained by PUF Phenotype images sent by the prover in order to be granted authentication. Increasing the confidence threshold has the effect of lowering false positives while potentially increasing false negatives. An ideal confidence threshold (minimum false negative rate) can be determined during enrollment, eliminating false positives when testing fraudulent data. This process (during authentication) is described in line 9 of the Authentication Phase shown in Algorithm 2. The ID check function applied in Algorithm 2 is shown in Algorithm 3. We experimentally verify the confidence value property for each classifier with our results in Section 3.3.3.

Algorithm 2: DPAN Authentication Phase

Authentication

Input: x: *PUF Phenotype Image*
Data: $\tilde{t}$: *Confidence threshold*

1 $\{UID|x\} \leftarrow$ Current device receives authentication request
`/* UID: Unique device identifier */`
`/* x: PUF Phenotype image */`
2 $Q \leftarrow$ ***Check***(UID)
3 **if** $Q == 0$ **then**
4 Abort authentication
5 **else**
6 $\{\hat{y}, S\} \leftarrow DPAN(x)$
`/* ŷ: Classification prediction */`
`/* S: Confidence score */`
7 **if not** $\hat{y} == UID$ **then**
8 Abort authentication
9 **else if** $S < \tilde{t}$ **then**
10 Abort authentication
11 **else**
12 Phenotype authenticity passed

3.2.7 DRAM Phenotype-based Authentication Network (DPAN)

We use a Convolutional Neural Network (CNN) in a popular model architecture consisting of a feature extractor, whose output is fed into a classifier as input. The feature extractor unit is a VGG16 CNN, which uses pooling layers to output lower-dimensional representations of the PUF response Phenotype images. This stage is effective as it enables the classifier to be trained on more appropriate PUF Phenotype image data features.

Algorithm 3: UID Check Function

Check

Input: ID: *Recieved Device Identifier*
Output: $\delta \in \{0, 1\}$: *ID found decision*
Data: DB_{UID}: *UID List*
q: Size(DB_{UID})

```
foreach DB_UID ∈ {UID_0, ..., UID_q} do
    if UID == ID then
        δ ← 1;
        return δ;
    else if not EOF then
        /* EOF: End Of File                                    */
        continue
    else
        δ ← 0;
        return δ;
```

3.2.7.1 Feature Extraction - Modified VGG16

We utilised a modified VGG16 CNN framework as our feature extractor unit due to its combined simplicity, efficiency and powerful capability as an image feature extractor in comparison to other well-known CNNs (AlexNet, ResNet) [78]. The architecture of the VGG16 used is shown in Figure 3.7. The network consists of 5 blocks of convolutional layers, which aggregate local information, each followed by a pooling layer performing dimensionality reduction, with 13 convolutional layers and 5 pooling layers. Each convolutional layer has 3×3 filters, and the pooling size is 2×2. In a standard (unmodified) VGG16 model, three dense (fully connected) layers perform the final classification of the features extracted from the initial convolutional and pooling layers. As the name suggests, fully connected layers consist of nodes, each with its connection to every node in the following layer. These layers enable extremely powerful function approximation, provided sufficient training data is available to fine-tune the layers. Naturally, these layers are the most data-intensive and account for the majority of storage requirements for a neural network, leaving the network at around 0.5GB. In our proposed scheme, however, we propose to replace these dense layers with lighter standard ML classifiers. As a result, we replace the three fully connected layers with a 1×1 average pooling layer to convert the original VGG16 from classification to only feature extraction. The original 200x220x3 input image is, therefore, output by the feature extractor as a 6x6x512 feature vector. Consequently, the lightweight VGG16 utilised by our scheme is only 57MB, reducing the storage and memory requirement by almost a magnitude of 10. This broadens the available applications of DPAN, enabling storage on many restricted devices, which is further discussed in Section 3.3.4. To compare the performance of the modified VGG16 with a standard VGG16, we provide benchmark classification results from an unmodified full VGG16 model with the dense layers intact as a feature extractor, which is discussed further in Section 3.3.

3.2.7.2 Tested Classifiers

The features output from the VGG16 model is utilised to train a standard ML classifier [12]. To determine the best-performing classifier for our application, we tested multiple popular classifiers on our data, each having passed through the same modified VGG16 CNN. As our data is labelled, we chose six supervised learning techniques to experiment with:

- **Decision Tree:** A Decision Tree is a simple supervised learning algorithm than can be used for regression and classification problems. As the name suggests, decision trees build tree-like structures based on rules formed given observations on features used during training. In the tree, leaves represent class labels and branches represent variations of features that lead to given class labels. Decision trees are extremely lightweight and easy to interpret as the tree can be formed visually to determine each rule based on any given input. It should be noted that it is not (reasonably) possible to determine a prediction confidence using a decision tree, which would be required for authentication. This is because multiple passes are required to determine class probability such as is possible with the random forest classifier. We included this classifier to benchmark the pure classification performance against the other chosen classifiers.

- **XGBoost:** XGBoost is an ensemble learning method and an implementation of gradient boosted decision trees, widely spread in applied ML. It is the most efficient and commonly used implementation of gradient boosting algorithms to date. XGBoost combines software and hardware effectively to produce superior results based on function approximation by optimizing specific loss functions and applying several regularization techniques. This has the effect of reducing the required computing resources and execution time. Key advantages include the handling of missing data automatically, parallel tree construction and continuation of training to further boost a trained model on new training data.

- **K-Nearest Neighbours:** The K-Nearest Neighbors algorithm (KNN) is a simple yet efficient classification method. The input data is classified to the most common class among its k nearest neighbors when plotted (k is a positive and typically small integer). The neighbors are taken from data belonging to the same class. The optimal choice of the value k is highly data-dependent: in general a larger k suppresses the effects of noise, but makes the classification boundaries less distinct.

- **Support Vector Machines:** Support vector machines (SVMs) are a set of supervised learning methods which can be applied to classification, regression and outlier detection problems. SVMs are one of the most robust prediction methods based on statistical learning frameworks, mapping training examples to points in space and optimising a plane based on the maximum distance between distance points of differing classes (known as the support vectors). New examples are then mapped into that same

space and are labelled with a class based on which side of the plane they appear. It is also effective in high-dimensional spaces and in cases where the number of dimensions is greater than the number of samples.

- **Random Forest:** Random Forest is a yet another ensemble method, more specifically bagging, which combines basic decision trees to become a robust classifier. As the random forest results are the average of the decision trees, it can significantly reduce the variance of the classification/regression.

- **Logistic Regression:** Logistic regression is a generalised linear model that can be used for classification and is also known as logit regression, maximum-entropy classification (MaxEnt) or the log-linear classifier. The probabilities representing the possible outcomes of an instance are classified via the logistic function.

This distribution of techniques represents a broad spectrum of standard classification techniques, including faster, simpler approaches such as K-nearest neighbours, alongside state-of-the-art ensemble methods such as XGBoost.

3.2.7.3 Training and Evaluation

We tested the modified VGG16 with each classifier on the test data and reported *Accuracy* and *F1-score* as our performance metrics. For the ablation study, we also compare our approach based on the lightweight modified VGG16 against a full VGG16 model trained with the dense layers present to classify groups of 3, 4 and 5 devices, respectively. This choice for the number of devices was limited by the number of unique DRAM DIMMs available when generating the dataset. Intuitions for the effects of increasing this number are discussed further in Section 3.3.5. Finally, during training, we performed K-fold ($K = 5$) cross-validation which ensured we could test our models across the entire distribution of the dataset as opposed to a standard train/test split, where the model only ever sees a specific portion of the dataset[‡].

3.2.7.4 Hyperparameter Tuning

Before finding our final model results, we performed hyperparameter optimisation for each classifier to enhance the performance of our models for our specific classification task. We performed a randomised grid search through a predetermined list of each available hyperparameter per classifier. During this phase, we used K-fold ($K = 5$) cross-validation as mentioned in Section 3.2.7.3 when training each model with each hyperparameter combination. Cross-validation aims to provide an estimate of model performance that balances bias and variance. Using too few folds (e.g., 2 or 3) can lead to high variance in the prediction,

[‡]**For the interested reader, a more detailed explanation for each evaluation metric is provided in Appendix D of the supplementary material.**

while using too many folds (e.g., 20 or more, depending on the size of the dataset) can increase computational cost without a significant reduction in bias. As a result, utilising five partitions has emerged as a highly common and effective compromise in ML practice and thus serves as the basis for our decision. For reproducibility, we provide the hyperparameters used for each classifier after tuning in Table 3.2. Additionally, we provide details on the grid used for our randomised grid search based on the ranges, also presented in Table 3.2 We performed each experiment using the Scikit Learn library in Python. For brevity, any model hyperparameters not included in Table 3.2 were left as the default.

Table 3.2: *Hyperparameter tuning values for classifiers*

Parameter	Description	Optimal value	Grid Search Ranges
Logistic Regression			
C	Type of regularization	1	0.001, 0.01, 0.1, 1, 10, 100
Max Iterations	Number of iterations	100	100, 200, 300, 500, 1000
K-Nearest Neighbour			
n_neighbors	Number of neighbours	9	1-30
Leaf Size	Min points in node	1	1-50
XGBoost			
Gamma	Min loss reduction required to make partition on leaf node	0	0, 0.5, 1, 1.5, 2, 5
Learning Rate	Rate of model learning during training	0.02	0.01, 0.02, 0.05, 0.1, 0.2, 0.3
Estimators	Number of trees	64	50-200
Max Depth	Max depth of a tree	3	3, 4, 5, 6, 7, 8, 9, 10
Min Child Weight	Min sum of instance weight needed in child	1	1, 5, 10
Subsample Ratio	Training data sample ratio prior to tree growth	0.6	0.6, 0.8, 1.0
Support Vector Machine			
C	Type of regularization	10	0.001, 0.01, 0.1, 1, 10, 100
Gamma	Kernel coefficient	0.1	0, 0.1, 0.5, 1, 1.5, 2, 5
Decision Tree			
Criterion	Function used to measure the quality of split	Gini	Entropy, Gini
Max Depth	Maximum tree depth	10	5, 10, 20, 50, 100
Leaf Min Samples	Min number of samples required at leaf node.	5	1, 2, 5, 8, 16, 32
Random Forest			
Estimators	Number of trees	396	100-500

3.3 Results and Discussion

In this section we evaluate the performance of each tested classifier in its classification performance using both the full VGG16 feature extractor and the modified lightweight VGG16

feature extractor. We also evaluate the confidence scores output for each model and discuss the ability of each classifier to identify legitimate phenotype inputs from fraudulent ones. Each classifier was tested for each three to five devices as discussed in Section 3.2.7.3. Tables 3.3 and 3.4 display the accuracy, F1 score, cross-validated mean accuracy and cross-validated standard deviation for each tested classifier.

3.3.1 Full VGG16 Feature Extractor Benchmark

The results in Table 3.3 demonstrate varying levels of success in the classification of each device between each classifier. The LR classifier displayed the best performance across most of the tests, with a classification accuracy of 95.7%, 96.8% and 91.7% for three, four and five devices, respectively. SVM also performed very well in each test, showing slightly better performance on five devices with an accuracy of 93.8%. While the RF and XGB classifiers performed comparably for fewer devices, each of these classifiers suffered more significantly for five devices, with accuracies down to 89.1% and 85.4%, respectively. The DT classifier demonstrated a vastly worse performance than each other classifier, showing the worst performance across each number of devices with classification accuracy as low as 65.1%. The highest achieved result for each number of devices was 95.7%, 96.8% and 93.8%. Intuitively, as the number of classes (devices) increases, the performance drops slightly on average due to the increased complexity of the models.

Table 3.3: *Classification Results with Full VGG16 Feature extractor*

Classifier	Accuracy			F1 Score			CV μ Acc.†			CV SD††		
	3x‡	4x	5x	3x	4x	5x	3x	4x	5x	3x	4x	5x
SVM‡‡	**0.957**	0.955	**0.938**	**0.957**	0.955	**0.938**	0.993	0.971	**0.987**	0.009	**0.025**	**0.023**
LR	**0.957**	**0.968**	0.917	**0.957**	**0.968**	0.917	**0.996**	**0.985**	0.982	**0.005**	0.030	0.033
DT	0.793	0.844	0.651	0.796	0.846	0.651	0.898	0.871	0.853	0.019	0.038	0.065
KNN	0.922	0.942	0.922	0.923	0.943	0.922	**0.996**	0.980	0.976	**0.005**	0.028	0.031
RF	0.948	0.955	0.891	0.949	0.955	0.890	**0.996**	0.984	0.978	**0.005**	0.029	0.035
XGB	**0.957**	0.948	0.854	**0.957**	0.948	0.854	**0.996**	0.984	0.978	**0.005**	0.029	0.035

† Cross-Validation Mean Accuracy; †† Cross-Validation Standard Deviation; ‡ Number of Devices;
‡‡ **DT**: Decision Tree; **KNN**: k-Nearest Neighbors; **LR**: Logistic Regression; **XGB**: XGBoost; **RF**: Random Forrest; and **SVM**: Support Vector Machine

3.3.2 Proposed Lightweight VGG16 Feature Extractor

Applying the lightweight VGG16 feature extractor showed a far greater performance across all classifiers in comparison to the full VGG16 feature extractor. The results shown in Table 3.4 indicate a strong ability for most models to accurately classify the origin of each PUF Phenotype in the testing set. Here again, the LR and SVM models performed the most consistently across the tests for each number of devices, correctly classifying almost all Phenotype images with between 97.9% to 98.3% accuracy. For the three device model, XGB performed exceptionally well, correctly classifying 99.1% of Phenotypes correctly. The XGB model did, however, suffer poorer results slightly with the increased number of devices. The RF classifier performed almost identically to the LR classifier; however, it performed the best for the higher number of devices, achieving an accuracy of 98.4% for five devices. Overall, the

LR and RF models were the highest overall performing classifiers. The confusion matrices for these two classifiers for each number of devices can be seen in Figure 3.8[§].

Table 3.4: *Classification Results with Lightweight VGG16 Feature extractor*

Classifier	Accuracy			F1 Score			CV μ Acc.			CV SD		
	3x	4x	5x	3x	4x	5x	3x	4x	5x	3x	4x	5x
SVM	0.983	0.974	0.979	0.983	0.975	0.979	0.983	0.975	0.959	0.035	0.049	0.075
LR	0.983	**0.981**	0.979	0.983	**0.981**	0.979	**0.998**	**0.995**	0.992	**0.004**	**0.010**	**0.010**
DT	0.957	0.968	0.927	0.957	0.968	0.927	0.989	0.974	0.973	0.022	0.048	0.048
KNN	0.983	0.968	0.974	0.983	0.969	0.974	**0.998**	**0.995**	**0.993**	**0.004**	**0.010**	**0.010**
RF	0.983	**0.981**	**0.984**	0.983	**0.981**	**0.984**	0.996	**0.995**	0.991	0.009	**0.010**	0.015
XGB	**0.991**	0.961	0.974	**0.991**	0.962	0.974	0.996	**0.995**	0.991	0.009	**0.010**	0.015

[§]**The confusion matrices for all experiments are provided in Appendices A and B**

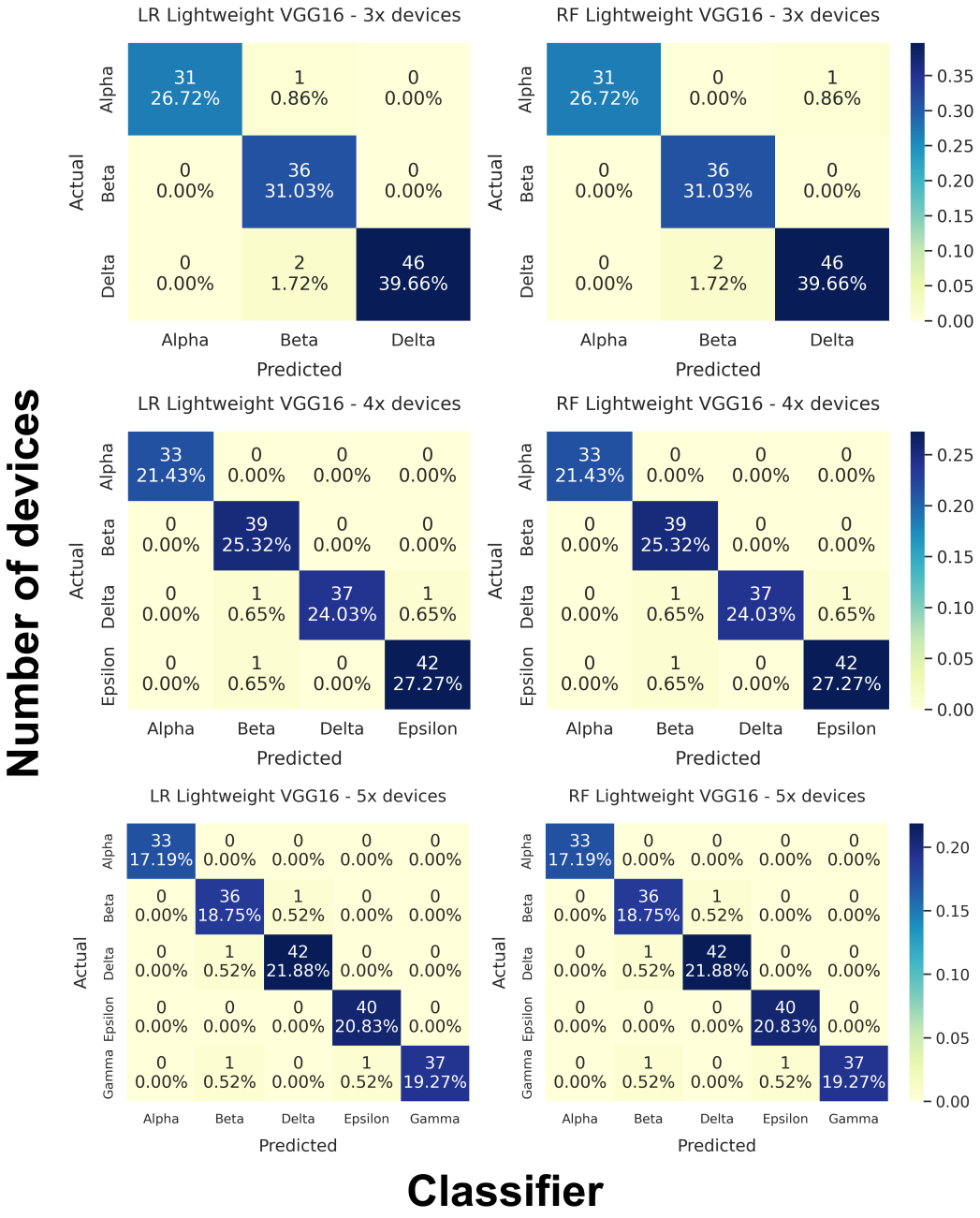

Figure 3.8: *Confusion Matrices of the Best Performing Classifiers for 3, 4 and 5 Devices*

3.3.3 Model Confidence

Table 3.5 shows the average confidence values for each correctly and incorrectly classified legitimate images, the maximum confidence over a set of ten fraudulent randomly generated images and finally the False Negative and False Positive rates determined by a tuned confidence value for each classifier. We see that both the SVM and RF classifiers perform

strongly when identifying fraudulent images. SVM performed exceptionally well with 4 to 5 devices, with maximum confidence outputs of 0.355 and 0.390 respectively. We determined fine-tuned confidence thresholds for each classifier determined by the maximum observed confidence scores for each classifier on fraudulent input images, shown in Table 3.6. Each value determines the confidence output required by images to be deemed authentic by each classifier (lower is better). When using a tuned confidence score to authenticate each test sample, SVM showed the best performance with no false positives for fraudulent samples and a maximum of 3.3% false negative rate across legitimate test samples. RF also performed strongly, however suffered from a slightly higher false negative rate of between 5.8% and 9.4%. While LR proved an excellent classifier of legitimate images, it performed poorly in identifying fraudulent images with a minimum confidence score of 0.900 (5 devices). XGB performed strongly in removing false positive results, however displayed a high false negative rate, which translates to poorer overall system performance.

Table 3.5: *Confidence Scores*

Classifier	μ True Conf.†			μ False Conf.††			Max μ Conf.‡			FN–FP (%)‡‡		
	3x	4x	5x	3x	4x	5x	3x	4x	5x	3x	4x	5x
SVM	0.978	0.952	0.927	0.800	**0.546**	0.535	0.748	**0.355**	**0.390**	**2.6–0**	**3.3–0**	**2.1–0**
LR	**0.992**	**0.979**	**0.979**	0.765	0.834	0.800	0.984	0.945	0.900	10.3–70	9.1–0	8.9–0
DT‡‡‡	-	-	-	-	-	-	-	-	-	-	-	-
KNN	0.981	0.985	0.974	0.778	0.778	0.667	0.667	1.000	0.778	7.8–0	7.1–20	8.3–0
RF	0.975	0.959	0.936	0.746	0.783	0.621	0.793	0.548	0.664	9.5–0	5.8–0	9.4–0
XGB	0.761	0.713	0.674	**0.707**	0.596	**0.450**	**0.395**	0.652	0.639	3.5–0	10.4–0	17.2–0

† Average confidence when classification is **correct**; †† Average confidence when classification is **incorrect**;
‡ Maximum confidence when classifying adversarial image; ‡‡ **FN:** False Negative; **FP:** False Positive;
‡‡‡ Decision Tree not applicable due to inability to determine confidence

3.3.4 Device Overhead

We finally performed an analysis of model execution for each classifier on a Raspberry Pi 3 Model-B to simulate performance in a resource constrained system (Figure 3.9). Table 3.7 provides the power overhead and execution time for a single image classification for each model alongside the storage requirements for each. The size of the modified VGG16 feature extractors were 57,589KB, 57,591KB and 57,593KB for the three, four and five device models respectively. The feature extractor size has been omitted from the table for ease of reading as each classifier uses the same lightweight feature extraction model. To determine the maximum power overhead, we monitored the power consumption of the Raspberry Pi in an idle state for three hundred seconds to establish a baseline power consumption of the device. During idle, the Raspberry Pi consumes on average 3.371 Volt Amperes, therefore we considered any value over this during model execution to be the power requirement overhead of for each model. In terms of maximum power consumption, no single classifier performed the best over each of the three, four and five device models, with RF demonstrating the lowest overhead for the three device classification, SVM for four devices and KNN for five devices. Execution time was more consistent, with the LR model performing the best in terms of execution times compared to the other models. RF overall had a higher required execution time for classification than the other classifiers, however this was not significant when considering

Table 3.6: *Fine-tuned confidence thresholds for each classifier*

Number of Devices	Confidence Threshold
Logistic Regression	
3x Devices	0.96
4x Devices	0.95
5x Devices	0.91
K-Nearest Neighbour	
3x Devices	0.68
4x Devices	0.80
5x Devices	0.79
XGBoost	
3x Devices	0.42
4x Devices	0.67
5x Devices	0.65
Support Vector Machine	
3x Devices	0.78
4x Devices	0.40
5x Devices	0.42
Random Forest	
3x Devices	0.83
4x Devices	0.65
5x Devices	0.68

its effectiveness in terms of accuracy and Phenotype authentication as discussed previously. Furthermore, training time is a factor for general consideration. Simpler classifiers such as LR, DT, RF and KNN have generally faster training times, wheras SVM and XGB classifiers require more time. This can have varying impact on a utilising system based on how often each model is required to be re-trained.

Table 3.7: *DPAN Device Overhead*

Classifier	Max Pwr (*VA*)			Size (*KB*)			Exec. Time (*Secs*)		
	3x	**4x**	**5x**	**3x**	**4x**	**5x**	**3x**	**4x**	**5x**
SVM	5.97	**4.78**	5.86	359	948	1,237	5.76	6.98	6.51
LR	5.84	6.00	5.80	13	17	21	**5.72**	6.18	**6.16**
DT	5.79	4.94	5.90	**2**	**3**	**3**	6.61	**5.96**	6.28
KNN	6.04	4.86	**5.72**	925	1,234	1,541	5.80	6.33	6.42
RF	**4.85**	4.94	5.78	276	648	1,241	6.79	7.21	7.43
XGB	5.76	5.94	6.20	108	170	261	5.67	6.68	6.26

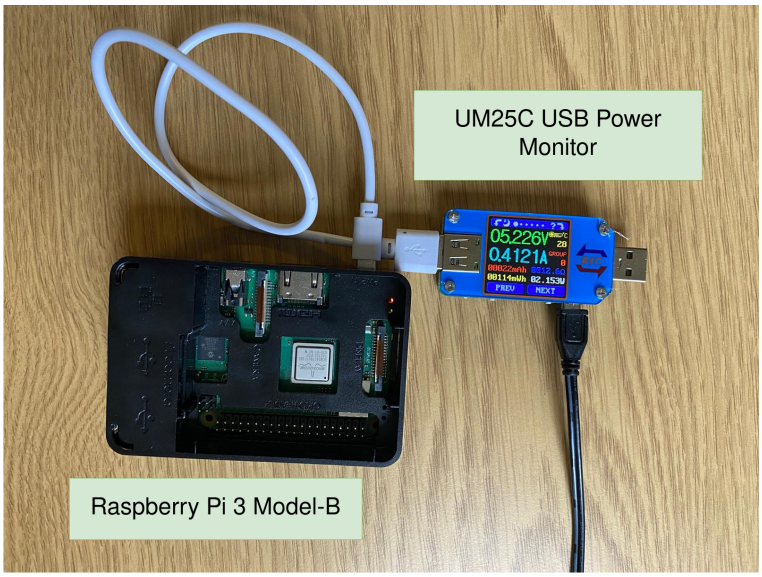

Figure 3.9: *Raspberry Pi Setup for Device Overhead Analysis*

3.3.5 Effect of Scaling Number of Devices

Intuitively, as the number of unique devices for authentication increases, so should the model complexity. This is demonstrated fundamentally in the increased model footprint, shown in table 3.7, whereby the model size in *KB* increases for all classifiers depending on number of devices. Additionally, it would seem intuitive to see a decrease in model performance as device numbers (number of unique classes) increases, unless the number of unique samples per device scales. In our experimentation, the number of unique samples remained the same for all training cycles per number of device, meaning the complexity of the model is scaling more significantly than the training data supplied. This likely explains the slightly higher performance of the three device models over the five device models. This result however is not universal, for example the RF classifier demonstrating the highest performance in classification accuracy for four devices and lowest for three, however only by a short margin. Intuition would suggest that as the number of devices scales further, the performance of the models would begin to significantly decrease unless significantly more unique samples were provided for each device during training. In the case of DRAM-PUFs, this would introduce a scalability ceiling whereby no more unique samples could be generated. Future work should

further explore the relationship between number of devices and number of samples when classifying DRAM-PUF responses.

3.4 DPAN Security Considerations

We address a set of possible security threats/concerns for the proposed system. We assume an adversary has physical access to the DRAM-PUF device, and is able to access and inspect the trained classification model stored on the device. As a formal protocol is out of the scope of the work in this chapter, we initially assume protocol-level security to be provided supplementary to the classification system to regard attacks such as replay, message tampering, denial of service, desynchronisation etc (as considered in Chapter 5).

Inference of PUF Behaviour An adversary may attempt to gain learn the PUF response data/entropy at some stage of the authentication phase. Usually, it would be necessary to consider an adversary looking to collect some helper data used for error correction in order to learn about the PUF CRP behaviour; in the proposed scheme, however, no helper data is required at any point to actively correct errors in the responses themselves, negating this security threat entirely. This leaves one threat to consider, being that of an adversary attempting to learn some information about the DRAM-PUF behaviour from the classification model itself. Due to the black-box nature of trained neural networks and the fact that the input and output are already publicly known, an inspection of the trained VGG16 or classifier would reveal no useful information to allow an adversary to infer the PUF behaviour. It should however be noted that knowledge of environmental conditions could provide an attacker with an amount of advantage when attempting to predict the PUF response bits. This is due to the fact that while the *distribution* of noise (location of randomly occurring bit flips) at varying conditions is random, the *amount* of noise is not. This phenomenon has been exploited in recent literature to detect the change in location of end devices via DRAM-PUF measurements, noting changes in amounts of noise [91]. While on a per-bit level this does not provide an attacker with assistance in guessing bits of the final response, it could potentially lead an attacker to generating a closer representation of the noise distribution of the final response if they gain knowledge of the operating conditions of the PUF, leading to an output which can fool the DPAN model. Future research should investigate the advantage gained from this attacker knowledge.

Response Guessing An adversary may attempt to brute force query the classification model to try to discover which images produce an acceptable confidence score to allow authentication. This would require a random guess of a response close enough to the hamming distance threshold, producing an acceptable confidence value when input to the model. This attack is extremely difficult as an attacker must guess the correct integer from 0-255 for each of 44,000 pixels in the image data, in addition to the guessed pixel value being correct enough for the model to extract comparable features to the legitimate images. We also experimentally verify this security property in Section 3.3.3, as when utilising an appropriate confidence threshold, DPAN is able to distinguish fraudulent responses to mitigate false authentication from adversarial images entirely. Finally, an attacker who knows the algorithms and class labels used in DPAN may attempt to collect some device responses to train their own new

classification models and learn some information to compromise the PUF and/or authentication system. This attack is also not economical for an attacker as training a new model is also a black-box process and does not enable an adversary to learn particular new inputs that could be used to deceive a legitimate model. An extremely similar, if not identical, classification model would need to be trained (which is difficult to confirm by the attacker) and brute force queried with fraudulent images to identify patterns that could potentially fool the real DPAN model.

PUF Modelling In a strong PUF-based system, noisy responses could theoretically be used to train a predictive model. Unlike other solutions, however, this model would need to produce not just the correct response but also noise in a distribution which is recognised as authentic by the DPAN classifier. The inclusion of noise in the transmitted responses is also an impediment to the construction of a predictive model for the PUF, though the degree to which this offers protection is an open question. Simple falsification is shown to be non-viable as experimentally verified in Section 3.3.3, where the CNN (with SVM, RF and XGBoost classifier) shows 0% false positives when authenticating responses consisting of real and fake responses. A sufficiently large set of acquired responses from the device set would potentially allow an adversary to duplicate the authentication model. This in itself does not allow for an attack, but it may be possible to use that model when training a predictive one which would allow for a direct attack.

Effects Poor PUF Uniqueness In our experiments, we utilise DRAM-PUF samples with a high uniqueness property. High uniqueness is an essential property of PUFs, such that individual PUFs can be uniquely distinguished from one another. When utilising CNN-based approaches for PUF authentication, PUFs with poor uniqueness may cause increased false-negative rates, whereby one individual PUF is misidentified as another, causing a rejection even when the originating device is legitimate. This emphasises the importance of utilising PUFs with a high uniqueness property for CNN-based authentication. Experimentation which investigates this effect is an interesting area for future research.

Replay Attacks As the scheme transmits responses over an open channel there is the risk of a replay attack, where an attacker transmits a previously recorded response when a challenge is re-used and thus passes authentication. In PUF schemes this is generally handled at the protocol by avoiding the re-use of previously issued challenges. There is a particular challenge in doing so in the proposed scheme as the noisy responses can't be matched to a 'correct' value, so the challenge used by any two devices must be communicated to every other device each time an authentication takes place and retained for comparison to future challenges. The suitability of existing protocol-level mitigations to schemes like the one proposed in this chapter is an interesting possibility for future work.

Poisoning Attack An adversary may attempt to send multiple noisy data to a device to impact the future performance of the model. This type of attack is not possible as poisoning attacks only effect online models which are continuously trained after deployment to remain up to date with continuous data and avoid concept drift. As training only occurs during

secure enrollment, DPAN will only output predictions to any input data and not retrain on it.

3.5 Summary

In this chapter, we have proposed an approach to strongly indicate the authenticity of noisy PUF identities based on a computer vision inspired ML classifier. By applying advanced noise tolerant classification schemes and the concept of a *PUF Phenotype*, it has been shown that PUF IDs can be classified and shown to be authentic with a high degree of confidence without the need for any knowledge of the underlying PUF structure, properties, noise characteristics, or environmental conditions. This allows for robust authentication without on-device error correction and without the need for privacy leaking helper data. The proposed approach has been verified on real DRAM Latency PUF data collected from commodity devices under a broad range of environmental conditions covering temperature and voltage fluctuation. In addition, six different classification methods were tested: XGBoost, RF, KNN, DT, LR, and SVM. Using this data classification accuracy and F1 scores of between 94% and 98% were achieved for varying number of supported devices respectively. We demonstrated that most tested classifiers can perform well with a tuned confidence threshold when identifying fraudulent and legitimate images, with the SVM and RF classifiers performing particularly well. It has been shown that using an appropriate confidence threshold, the risk of determining a high confidence for false responses can be eliminated while still detecting true responses on the first try in most cases. The need for occasional retries due to this is still significantly less PUF usage than would be required for high quality on-device error correction. As in this chapter we tackle a PUF-level authentication methodology for purely DRAM-PUFs, we encounter restrictions regarding the amount of unique CRPs supported by DRAM-PUFs. To prevent replay attacks, unique CRPs would need to be single-use only, which impacts the scalability of the scheme as devices would need to be recalled after all CRPs are exhausted. It is therefore necessary to investigate how DRAM-PUFs can be deployed with the strengths of Strong PUFs through CRP space to enable authentication.

Chapter 4

A Generic Obfuscation Framework for Preventing ML-Attacks on Strong PUFs through Exploitation of DRAM-PUFs

In this Chapter, we propose a generic framework for securing Strong-PUFs against ML-MA through obfuscation of challenge and response data by exploiting a DRAM-PUF to supplement a One-Way Function (OWF) which can be implemented using the available resources on an FPGA platform. Our proposed scheme enables reconfigurability, strong security and one-wayness. We conduct ML-MA using various classifiers to thoroughly evaluate the performance of our scheme across multiple 16-bit and 32-bit Arbiter-PUF (APUF) variants, showing our scheme reduces model accuracy to around 50% for each PUF (random guessing) and evaluate the properties of the final responses, demonstrating that ideal uniformity and uniqueness are maintained. Even though we demonstrate our proposal through a DRAM-PUF, our scheme can be extended to work with memory-based PUFs in general.

4.1 Introduction and Related Work

Several schemes have been proposed that aim to fortify the security properties of Strong PUFs. The fundamental core of all ML-MA is that the relationship between a PUF challenge and the matching response is the PUF function itself. Given a training set of known CRPs, it is possible to infer a model that predicts the PUF function's behaviour for all challenges, including those not in the training set. Therefore, anti-modelling countermeasures aim to obfuscate this relationship and, by extension, the PUF function. Ideally, this would render ML-MA impossible, but in practice, many methods aim to raise the modelling complexity enough to make attacks impractical in the field. Obfuscation methods can mainly be categorised in two ways with subtle differences: structural non-linearisation and CRP obfuscation. Structural non-linearisation involves designing PUFs where the actual PUF function and challenge-response relationship are less linear in structure, thus increasing the modelling complexity [29, 45]. CRP obfuscation aims to obscure the challenge and responses

via masking, hash functions, etc., so CRPs cannot be used to infer the PUF function unless the adversary can also reverse the obfuscation function [24, 28, 80].

In 2004, Gassend et al. introduced the FF-APUF, whereby the results of early PUF stages are fed-forward to several challenge inputs further along, reducing the linearity of the PUF [29]. While in practice, this PUF showed strong resistance to Linear Regression (LR) based ML-MA, Alkatheiri et al. demonstrated a successful attack using a Multi-Layer Perceptron (MLP) to model the FF-APUF [1]. In 2007, Suh et al. proposed the first PUF to include CRP obfuscation in the XOR-APUF, where the output of several unique APUFs is XOR'ed together to obscure the mapping of each APUFs responses against the same challenge [80]. This additional logic (and added resistance to ML-MA) came at the expense of reduced PUF reliability and increased hardware overhead. Ma et al., Miskelly et al., and Cui et al. proposed the use of lightweight single-cell Weak PUFs as an alternative method of CRP obfuscation [56] [67] [14], where each CRP bit is XOR'ed with the response of a single bit Weak PUF. The hardware overhead of this approach is minimal, but in practice only reduces the prediction rate of advanced attacks with large CRP training sets to around 80%. This gives the adversary a significantly increased chance of achieving a collision, even if it prevents fully reliable prediction.

Gassend et al. proposed a Controlled PUF, whereby hash functions are utilised to restrict access to CRPs [28]. The combination of multiple hash functions and error-correction code (ECC) incurs a substantial hardware requirement, making it difficult to justify the PUF over traditional cryptographic methods. Ye et al proposed an RPUF in [92], whereby each challenge is randomized before input to the PUF. This scheme, however showed insufficient resilience against ML-MA with most attacks providing above 70% prediction accuracy. Subsequent attacks increased this to above 90% [16]. In [24], Gao et al presented PUF-FSM, where ECC is replaced with a finite-state machine. However, the hash logic still required by the scheme ensured the hardware overhead was not sufficiently low and the selection for reliable responses in the CRP set provides enough information leakage to mount a successful attack as in [16]. Dubrova et al proposed the CRC-PUF, utilising a circuit based on Cyclic Redundancy Checking to perform a transformation of the PUF input to increase the difficulty of ML-MA [20]. While compact and generically applicable, this solution's efficacy is unclear. Only one ML-MA method was tested experimentally, LR, and even then a prediction rate of 75% per bit was achieved. It is also not a complete scheme as it relies on the generation of a polynomial to configure an LFSR each time the PUF is used, but no specific method for generation or handling at the protocol level is proposed and this is not considered in the resource usage.

Recently, Zhang et al proposed an obfuscation scheme for Strong PUFs that utilises prestored stable PUF responses for obfuscation after enrollment [95]. A TRNG is used to select randomly from a set of keys to obscure CRPs using XOR operations, and when a number of CRPs can be collected by an adversary, the set is updated. While this scheme enabled strong resilience against ML-MA and lower hardware overhead than similar schemes, it relies on storing key data in NVM, enabling adversaries with physical access to gain access to the secret data, which would ultimately compromise the security of the PUF. Zhang et al. also proposed a scheme which uses a structure that can operate as three different kinds of PUF (arbiter, ring oscillator and bi-stable ring), called CT-PUF [96]. In order to reduce the linearity between challenges and responses, the CT-PUF acts as only one of these PUFs

to any given challenge, but the challenge itself is also obscured by being fed through an arbiter PUF. The CT-PUF structure is more area-efficient than other proposals as the single circuit provides all three PUF functionalities. It achieves very good, but not perfect, ML-MA resistance with prediction rates of just over 60%. However, it is not a generalised scheme that can apply to any strong PUF. Additionally, while it is impressively compact, it will be shown in this work that even greater area reductions can be achieved through re-use of existing components.

4.1.1 Comparison

Ideally, there are several desirable properties (DP) for a PUF obfuscation scheme to exhibit such that the additional device overhead can be justified over simply adopting a PUF. When considering the threat model (described in detail in Section 4.2.3) of an attacker with physical access to a device and modelling capabilities, we outline a set of five key requirements which cover this assumption:

- **DP1: Minimal Area Requirement**:- PUF obfuscation requires additional logic and/or operations on top of the PUF circuit itself. Thus, to keep the obfuscated PUF lightweight and applicable to constrained systems, the obfuscation mechanism must use the lowest amount of system resources possible. Additional circuitry should be minimised for power and area, computation should be kept to a minimum, and existing system resources should be repurposed as much as possible.

- **DP2: ML-MA Secure**:- The scheme must provide sufficient non-linearity between PUF inputs and outputs (CRPs) such that an attacker with knowledge of part of the CRP set cannot mount a modelling attack using machine learning that results in a prediction rate significantly above 50% for CRPs not in the known set.

- **DP3: No NVM Requirement**:- A common requirement in authentication systems and in previously proposed PUF obfuscation schemes (such as [95]) is an NVM to store secrets or helper data on the device. Having a secret or data which reveals compromising properties of the system constantly present on the device during operation is not ideal and creates an obvious point of attack for adversaries. Further, in a PUF context it weakens a key property of the PUF, that the PUF secrets exist only when being used. A PUF obfuscation scheme, therefore, should try and avoid an NVM requirement and in general avoid the need for fixed additional secrets or helper data which contains information about the PUF properties. If supplementary data must be used, it should not give the adversary any information about the PUF properties or behaviour.

- **DP4: Reconfigurable**:- Ideally, in addition to passively preventing ML-MA by raising the difficulty of modelling, an obfuscation system should contain a mechanism for reconfiguration. i.e. changing the system behaviour such that using the same challenge twice between different configurations exhibits different final outputs. This provides an element of active countermeasure. If an attack is suspected, the PUF can be reconfigured, and a model trained to predict the previous configuration must be retrained.

- **DP5: Generic**:- The scheme should be designed separately from any given PUF design, such that designers can use any arbitrary PUF in the scheme and produce the

same enhanced security. This modularity ensures that the obfuscation scheme can be tested on future-developed PUFs and implemented to secure any strong PUF which a given manufacturer desires. This enables the use of previously designed PUFs with specific desirable properties, such as high reliability in the case of shorter length Arbiter PUFs, which are vulnerable to ML-MA without additional obfuscation logic [36]. A truly general scheme gives the designer the freedom to tailor the properties to the application's needs.

Table 4.1 compares the relevant works presented against the identified desirable properties. Each related work was chosen as relevant by the following criteria:

1. Utilises a form of Strong PUF.
2. Includes logically obfuscating steps both before and after Strong PUF input/output interfaces.
3. Primarily built in hardware - software level obfuscation is not included.

Factors in Table 4.1 listed as 'Inconclusive' are due to implementation details having not been provided by the original authors. The key distinguishing factors are DP1 and DP5. The proposed scheme is generic and enables the use of any strong PUF in a modular way, while other schemes are limited in this regard. The proposed scheme achieves *DP5* by completely separating the chosen PUF from the rest of the obfuscation logic, such that any strong PUF output is fully obfuscated with the lightweight proposed one-way function (described in detail in Section 4.2.1). While some other schemes do achieve this generic quality, they end up using more resources than the design proposed in this work. The proposed scheme achieves this reduction (*DP1*) by utilising existing resources on the device in the form of memory PUFs, which enable entropy generation in runtime without additional logic of use of permanently storing data in NVMs. To the best of our knowledge, the schemes included for comparison include all strong PUF-based schemes available in the literature at the time of writing which provide results from real implementation on FPGA. Comparison with PUF-simulation based work in this area, e.g. [98], would not be appropriate.

Table 4.1: *Comparison of the Proposed Scheme Against Similar Schemes*

Scheme	*DP1*	*DP2*	*DP3*	*DP4*	*DP5*
Controlled PUF [28]	X	∼	X	✓∗	✓
PUF-FSM [24]	X	X	✓	✓∗	✓
Set-based [95]	X	✓	X	✓	X
CRC-PUF [20]	∼	∼	✓	✓	✓
CT-PUF [96]	X	✓	✓	✓	X
Proposed Scheme	✓	✓	✓	✓	✓

✓: Yes; X: No; ∼: Inconclusive/Not Explicitly Evaluated
∗ If hash function used is reconfigurable
$DP1$: **Minimal Area** $DP2$: **ML Secure** $DP3$: **No NVM Required** $DP4$: **Reconfigurable** $DP5$: **Generic Framework**

4.1.2 Motivation

The major issue with Strong PUF obfuscation schemes is not in devising the obfuscation methods - any number of known hard problems can be applied to increase the modelling complexity to an infeasible degree. The key problem is: how to increase complexity such that the lightweight nature of the PUF concept is retained? If the obfuscated system requires many aspects of traditional cryptography, most of the benefits of a Strong PUF over purely cryptographic functions are lost. There is still the fact that it is rooted in hardware, however this benefit is not unique to Strong PUFs. The key goal, then, is to find the method(s) which add the most complexity for the lowest cost. For most schemes, there are three inescapable overheads in addition to the PUF itself:

(a) Some non-reversible function or mask to de-correlate challenges from responses.

(b) A fixed matrix or secret can be used in that function and an NVM to keep it in. This can sometimes be avoided depending on the nature of (a), but for very lightweight functions, this is generally a requirement.

(c) Strict error correction is necessary due to the error-amplifying nature of most applicable functions and inherent PUF noise.

As error management is always going to be necessary in PUF-based systems (be it explicit or ML-based as demonstrated in the previous chapter), the two targets for overhead reduction are the masking function (a) and the fixed secret/NVM (b), however schemes which avoid these have less than ideal results. Schemes which rely on more complex PUF structures alone, avoiding (a) and (b), have thus far largely proven vulnerable to more sophisticated ML-MA. Schemes which rely on very lightweight masking instead of a hash or one-way function, avoiding (b), only partially impede attacks. Steady progress has been made in reducing the footprint of masking functions, yet the overall footprint remains higher than is desirable, especially in systems using NVMs (which are relatively expensive, bit-for-bit). NVMs also present an obvious target for hardware-level attacks as they typically present a single point of failure for the obfuscation system.

Something less explored in this context is that PUFs are a type of 'memoryless' key storage mechanism, often used in place of NVM. For most PUFs, this results in no net gain in cost, as the PUF would need as much hardware as an NVM while being less stable. However, considering a Software PUF based on a volatile memory becomes a much more interesting proposition. It is safe to assume most systems will have at least one volatile memory. If this memory may also be exploited as a PUF, a concept which has been demonstrated on both SRAM [39] and DRAM [40, 41, 68], then it can be used as a matrix generator to remove the need for NVM entirely. If such a Memory PUF can be operated during system runtime as is the case for [41] and [68], it also provides the advantage of the fixed secret only existing when needed, rather than being constantly present on the device. Further, a system's main memory will naturally be much larger than what could be justified for NVM in a single subsystem, allowing for a larger fixed secret than would otherwise be feasible. In addition to the issue of resource use, there is a practical consideration around the diversity of PUF designs proposed to date. An obfuscation scheme based around a particular Strong PUF is helpful but not as useful as a generic scheme into which any existing Strong PUF IP can be inserted.

However, a generic scheme requires more rigorous testing because the modelling complexity of the PUF function is not fixed. Scheme resistance to one ML attack for one PUF type does not necessarily extrapolate to equal resistance to any given ML attack for any given PUF type. This is less of a concern for rudimentary attacks but more crucial when considering adversarial learning, deep neural nets, evolutionary strategies, and similar advanced ML techniques, which are well suited to fitting for highly complex functions. Ideally, a scheme should be both generic and tested experimentally against a broad range of ML techniques for various complex PUFs.

4.1.3 Contributions

To provide a solution to the limitations identified in the current literature, we propose in this chapter a generic DRAM-PUF-based obfuscation scheme for Strong PUFs. The main contributions of this chapter are as follows:

- A generic PUF-based obfuscation scheme for any Strong PUF, experimentally verified using multiple 16-stage and 32-stage Arbiter-based PUF variants simulated in software. To the best of our knowledge, we are the *first* to consider a modular scheme with regard to Strong PUF where an adversary is considered to have full knowledge of the utilised underlying scheme.

- A novel modified One-Way Function (OWF) to enable strong resilience to ML-MA and enhanced one-wayness to our scheme, without impacting ideal Uniqueness and Hamming Distance properties. In this regard, we utilise DRAM-PUFs for OWF configuration, having the effect of improved security with reduced hardware footprint.

- An evaluation of our scheme against well-known Machine Learning Modelling Attacks against strong PUFs, including a novel set of supervised and unsupervised classifiers tailored for our scheme, showing a strong capability to resist ML-MA.

- Synthesis of our OWF on a Zynq-7000 FPGA and provide a comparison of our proposed scheme for hardware overhead and power consumption against comparable schemes.

- We provide all source code and data used for our experiments, open access for the research community (Section 4.6).

4.1.4 Chapter Organisation

The remainder of this chapter is organised as follows. Section 4.2 introduces preliminary information regarding Memory-PUFs, One-Way Functions and notation/technical concepts used throughout the chapter. Section 4.3 provides a detailed description of the operation of the proposed scheme. Section 4.4 explains the experimental methodology undertaken to evaluate the proposed scheme. Section 4.5 then provides a breakdown and analysis of our experimental results, including a comparison against similar schemes, and finally, in Section 4.6, we provide a conclusion of this work.

4.2 Preliminaries

In this section, we outline some preliminary concepts pertinent to the proposed scheme and evaluation methods.

4.2.1 Custom Lightweight One-Way Function

In this work, we modify the compression function given in [22] (it is the improved variant of the function proposed in [3]) to design our One-Way Function (OWF). Here, we use our own truly random generated H matrices by using DRAM-PUF data instead of a random quasi-cyclic matrix in order to ensure optimal security. In [22], it is reported that this construction is not suitable for memory-constrained environments due to the storage requirements for the H matrices. When utilising the DRAM-PUF for storage and generation of H, this constraint is removed (discussed further in Section 4.2.1.1). In the original construction of the compression function, as described in [3], denoted F, takes s bits of input data, and uses a random $r \times n$ binary matrix to obtain r bits of output data. F consists of XOR operations and simply computes the syndrome of the split parts (i.e., s bits of data are split into blocks of w, and then w columns of H are XORed). The inverting the F is a Syndrome Decoding problem that is NP-complete [7]. It proves that when you choose the appropriate parameters, this compression function can be used as an OWF.

In [22], the authors improved the previous original construction [3] by using a quasi-cyclic matrix H instead of the generic random matrix to increase the overall efficiency. But still, this improved compression function does have its limitations, for example, the size of H is still a limiting factor for the efficiency of the compression function. Furthermore, the quasi-cyclic codes cannot guarantee complete randomness [22]. Therefore in this work we focus on how we can generate a truly random small binary matrix H of 8,192 bytes (note that here the parameters $r = 64$, and $n = 1024$) with appropriate w parameters, without any security loss for designing our OWF.

4.2.1.1 H-Matrix Entropy and Resourcefulness

A basic approach to the the H matrix requirement in hardware would be to generate a cryptographically significant matrix (through an arbitrary PRNG/TRNG) during device enrollment and permanently store it in a form of NVM to be accessed when required. This, as mentioned previously, creates a significant weakness for a PUF device, where it is assumed an adversary has physical access and could read H in this scenario which would entirely undermine the security of the PUF scheme, not to mention the significant hardware overhead incurred for each bit of required NVM. For these reasons, we focus on the benefits of exploiting Memory-PUFs which are already suitable to utilise available device resources to generate large amounts of high entropy data (discussed further in Section 4.4.4).

By modifying the idea given in [22], the steps for the proposed OWF can be written as follows:

1. Select number of blocks w,
2. Split m bits of input data into w blocks - if padding is required, pad now

Algorithm 4: One-Way Function (OWF)

Input: $M : R_{Origin}$
Data: w: *Block number*
Output: C_{Final}

$M' \leftarrow M \| pad(M, w)$
$m_1 \| \cdots \| m_w \leftarrow$ ***parse*** (M')
`/* where` $|m_i| = b$`-bit for all` $1 \leq i \leq w$ `and` ***parse*`()` `is a function that splits` M' `into` w `blocks */`
$x_i \leftarrow \langle m_i \rangle_b$
`/* where` x_i `is the integer representation of` b`-bit bit stream` m_i `*/`
for $i \leftarrow$ *1 to* w **do**
  $H_{x_i} \leftarrow$ ***pick***(H, x_i)
  `/* where` ***pick*`()` `is a function that picks the corresponding column in the` H `matrix at position` x_i `*/`
  $C_{Final} \leftarrow (C_{Final} \oplus H_{x_i})$
return C_{Final}

3. Convert each of the blocks from binary to integer $(x_1, x_2 \cdots, x_w)$,
4. Pick the corresponding column in the H matrix at position x,
5. Return $C_{Final} = H_{x_1} \oplus H_{x_2} \oplus \cdots \oplus H_{x_w}$

In our OWF, according to the chosen value of w, m needs to be padded. We check whether m is divisible by w, if it has no remainder the padding is not necessary, otherwise it is required. For the padding, it is inspired by PKCS#7 [43], however we made some modifications: the number of required padding bits is to be calculated with the following formula:

$$N_{padding} = ([(q_1 + 1) \times w] - m) \tag{4.1}$$

where q_1 is the quotient of m and w. Then, the result is converted to a 4-bit binary number, and we repeat it y times. Let q_2 be the quotient of $N_{padding}$ and 4, if $N_{padding}$ is divisible by 4 without remainder, y is equal to q_2, otherwise, will be equal to $(q_2 + 1)$. As a final step, we drop the last $(4 * y - N_{padding})$ bits to get the padded input data. For example, let w be 7, in here we need $6\,(= [(9 + 1) \times 7] - 64)$ bits padding, the 4-bit binary representation of 6 is "0110", then we repeat "0110" $2(= 1 + 1)$ times as "01100110". We only need 6-bit padding so we remove the last $2(= (4 \times 2) - 6)$ bits, finally, the pad will be "011001". When evaluating the hardware cost of the OWF function, in theory, splitting m has no cost. Here, the last step costs only $[(w - 1) \times m]$ binary XORs.

According to Algorithm 4, the concatenation of bit strings M and $pad(M, w)$ is denoted by $M \| pad(M, w)$ where $M \in \{0, 1\}^{64}$ is an 64-bit string and $pad()$ is a padding function. When x is an integer, we write $\langle m_i \rangle_b$ to represent its corresponding b-bit binary representation.

4.2.2 Notation

Uniformity

Uniformity ensures that every possible output of a cryptographic process is equally likely, which is an essential property for secure systems. In the context of PUFs, under the different challenges, the uniformity metric gives the distribution of '0's and '1's in PUF responses. Uniformity is formally defined as the following:

To define the necessary components when determining uniformity, we provide the following index:

1. Random Variable (X): A variable that can take on various values, each with a certain probability.
2. Finite Set (S): A set with a limited number of elements. For example, the set of all possible outputs of a cryptographic function.
3. Uniform Distribution: A type of probability distribution where every outcome in the finite set S has an equal chance of occurring.
4. Probability (Pr): A measure of the likelihood that a given event will occur. In this context, it refers to the likelihood that X takes on a specific value.
5. Cardinality ($|S|$): The number of elements in the set S.

Therefore, a random variable X taking values in a finite set S is said to be *uniformly distributed* if each element in S has an equal probability of being selected. This can be mathematically expressed as:

$$\Pr(X = x) = \frac{1}{|S|} \quad \forall x \in S \tag{4.2}$$

where Pr denotes the probability, X is the random variable, and $|S|$ represents the cardinality (number of elements) of the set S.

Uniformity can thus be defined using the following notation:

$$\Pr(X = x) = \frac{1}{|S|} \tag{4.3}$$

Now we consider a PUF that produces an output from a set $S = \{0, 1\}$. If the function is uniformly distributed, the probability of getting a 0 or a 1 is equal, i.e.,

$$\Pr(X = 0) = \Pr(X = 1) = \frac{1}{2} \tag{4.4}$$

4.2.3 Threat Model

This work is based on a threat model which assumes a *remote adversary* with the technical ability and resources to carry out advanced ML-MA. Specifically, the following assumptions are made:

1. The adversary is remote, or has limited physical access to the target device. While physical level attacks against PUFs are a factor of consideration they are not the primary motivator of this work.

2. The adversary has a goal of being able to predict the final output, R_{Final}, for any given challenge, C_{Origin}.

3. Any initial challenge, C_{Origin}, and corresponding R_{Final} used by a device are available to the adversary via eavesdropping. It is assumed the attacker can always acquire a moderately sized subset of the possible CRPs for any given device.

4. The internal feedback challenge and response, C_{Final} and R_{Origin}, are not available to the adversary and cannot be directly measured. The adversary may attempt to predict them as a step in the ML-MA process but has no way to verify this except through changes to the predicted R_{Final}.

5. The adversary has full knowledge of the scheme structure.

6. The adversary has full knowledge of the OWF structure and can execute a copy of it at will.

7. The adversary has full knowledge of which PUF designs have been used and the mechanisms they employ.

8. The adversary can send falsified challenges to the scheme as a whole but cannot bypass the scheme to query either the internal SPUF or Memory-PUF individually.

It is important to note that the assumption of restricted internal access to the Strong PUF (maintaining secrecy of R_{Origin}) is key to the security of the scheme and is worth particular consideration. An attacker gaining access to R_{Origin} would enable a modelling attack on the underlying Strong PUF, meaning with knowledge of C_{Origin}, H and the OWF, an attacker could relatively simply calculate the stages of the obfuscation scheme and predict R_{Final} bits. This would require an extremely accurate model of the PUF however, as a single bit predicted incorrectly would avalanche through the function, causing incorrectly predicted R_{Final} bits. Regardless, the attacker would have a strongly increased overall predictive advantage.

Accessing the response interface of the PUF lends arguments to be made for the inclusion of side channel analysis, hardware tampering, and/or fault injection as would be possible for a sufficiently advanced adversary with physical access. While not to be dismissed, experimental analysis of these attacks is beyond the scope of this work which is focused on the ML-MA threat. Some preliminary discussion of the implication of these other attacks for the proposed scheme is given in Section 4.5.2 with further exploration likely to form the basis of future work.

4.3 Proposed Scheme

For an adversary to collect a subset of possible CRPs and accurately model the PUF behaviour, there must exist a reasonably strong correlative relationship between the initial challenge and final response data. Based on this assumption, we aim to complicate the

mapping between initial challenges and final responses collected by an adversary such that resilience against ML-MA is sufficiently enhanced. Additionally, we consider the requirement to maintain a small enough hardware footprint to enable a scheme that can be reasonably deployed on edge devices.

We, therefore, present a generic obfuscation scheme where the Strong PUF element is considered modular, such that various Strong PUF implementations may be used interchangeably to generate hardware-centric tokens for further processing. To limit the overall footprint of a PUF obfuscation scheme it is desirable to enhance entropy using existing resources where possible. We achieve this in two ways: Firstly, while Strong PUFs on their own are susceptible to ML-MA, there is still an amount of entropy which is generated from the manufacturing variation of the PUF for each CRP. We exploit this by introducing a self-feedback feature where initial Strong PUF responses are used to create a new obfuscated challenge for the same Strong PUF. This contributes a baseline increase in modeling complexity for a remote adversary, as they will not have access to the internal feedback challenge. It adds very little hardware and is a generic solution which can be applied to any strong PUF. Further, it provides some flexibility in that depending on the stability of the PUF and acceptable level of resource consumption the number of feedback rounds can be increased or decreased.

In itself this increase in complexity is not sufficient to prevent ML-MA, however. Therefore, we also propose to use a low-cost OWF (as described in Section 4.2.1) in the feedback loop, such that the relationship between the origin challenge and the internal challenge becomes non-reversible and extremely difficult to predict without knowledge of a matrix, H, and the internal PUF response, R_{Origin}. In addition, we propose to exploit a second PUF - a Memory-PUF - as a matrix generator to supply the OWF with cryptographically significant matrices for configuration.

Generating matrix data in this way has key benefits. The OWF requires H to be a large random matrix, which would typically require permanent storage on-device. This both increases the scheme footprint and provides an obvious target for hardware level attacks. By offloading the synthesis and storage of H data to a Memory-PUF, both limitations can be mitigated. In terms of cost, the Memory-PUF can generate very large matrices with the necessary properties from its own hardware. In the case of the exemplary DRAM-PUF used in this work a significant proportion of the memory tested was usable for matrix generation, giving a total matrix space in the order of 100s of MB per GB of total DRAM (see section 4.4.4 for further details). An equivalently sized fixed matrix with NVM storage would incur a substantial hardware and processing cost.

In combination these measures provide a very significant increase to the modeling complexity while using minimal additional circuitry. The specifics of the scheme are given in the following section, with tests of attack resiliency, resource consumption, and security analysis provided in Sections 4.4 and 4.5.

4.3.1 Generic PUF Obfuscation

We present our proposed generic obfuscation scheme, the entire process of which is depicted in Figure 4.1 and is described with the following numbered steps:

Response format: A binary vector of length i.

Challenge format: The SPUF is assumed to generate 1 bit responses to x bit challenges, where x is typically the number of SPUF stages. The OWF requires a fixed 64-bit input, meaning $64x$ challenge bits are required for one execution of the OWF. This will produce a 64-bit output, which can be split into $64/x$ feedback challenges to produce $64/x$ bits of the final response. This must be repeated until the desired response size, i, is reached. Therefore the expected challenge C will consist of the full set of required SPUF challenges, C_{Origin}, concatenated with the Memory-PUF challenge C_{Mem}. C_{Origin} will be of length $(64x)*(i/(64/x))$ bits.

① First, the challenge data C is received by the device and split into the SPUF challenge C_{Origin} and Memory-PUF challenge C_{Mem}. The first $64x$ bits of C_{Origin} are taken as C^i_{Origin}.

② C^i_{Origin} is passed to the SPUF to generate the internal response, R^i_{Origin}.

③ Error correction is performed on R^i_{Origin} to remove noise. PUFs contain an inherent degree of random noise largely influenced by environmental condition variation. Therefore, error correction is performed on the PUF output to remove this noise and generate a stable output. The scheme does not specify a correction method as the optimal approach will vary depending on the particular properties of the chosen PUF (an example however is provided in Section 4.5.3 to demonstrate hardware overhead).

④ If $i <= 64$ (i.e., the scheme is in the internal feedback stage) R^i_{Origin} is appended to a secure 64-bit register where the full R_{Origin} is assembled.

⑤ The secure register waits for a full 64 bit R_{Origin}. This register must be carefully secured as whole or partial leakage of R_{Origin} invalidates the ML-MA countermeasures.

⑥ The completed R_{Origin} is passed to the OWF, which now requires the configuration matrix, H.

⑦ The Memory-PUF challenge C_{Mem} is passed to the Memory PUF controller, which prepares the memory segments indicated in C_{Mem} for PUF use.

⑧ The Memory-PUF generates a preliminary (noisy) response matrix.

⑨ Error correction is performed on the Memory-PUF output to produce the final de-noised matrix, H. Provided there is sufficient memory space available, this is carried out also within the memory (to store a copy for majority voting over repeated measurements).

⑩ H is used to configure the OWF, which has already been passed R_{Origin}. The 64 bit output of the OWF is the internal challenge, C_{Final}.

⑪ C_{Final} is divided into $64/x$ vectors of length x, which are fed sequentially as challenges into the SPUF.

⑫ The output bits are again error corrected to remove noise, then the R_{Final} bits are appended to the final response R.

(13) Finally, i is re-initialised to 0 and the process is repeated, using the next $64x$ bits of C_{Origin} as C^{i+1}_{Origin}. This continues until the required size of R is reached, after which R is returned to the verifier.

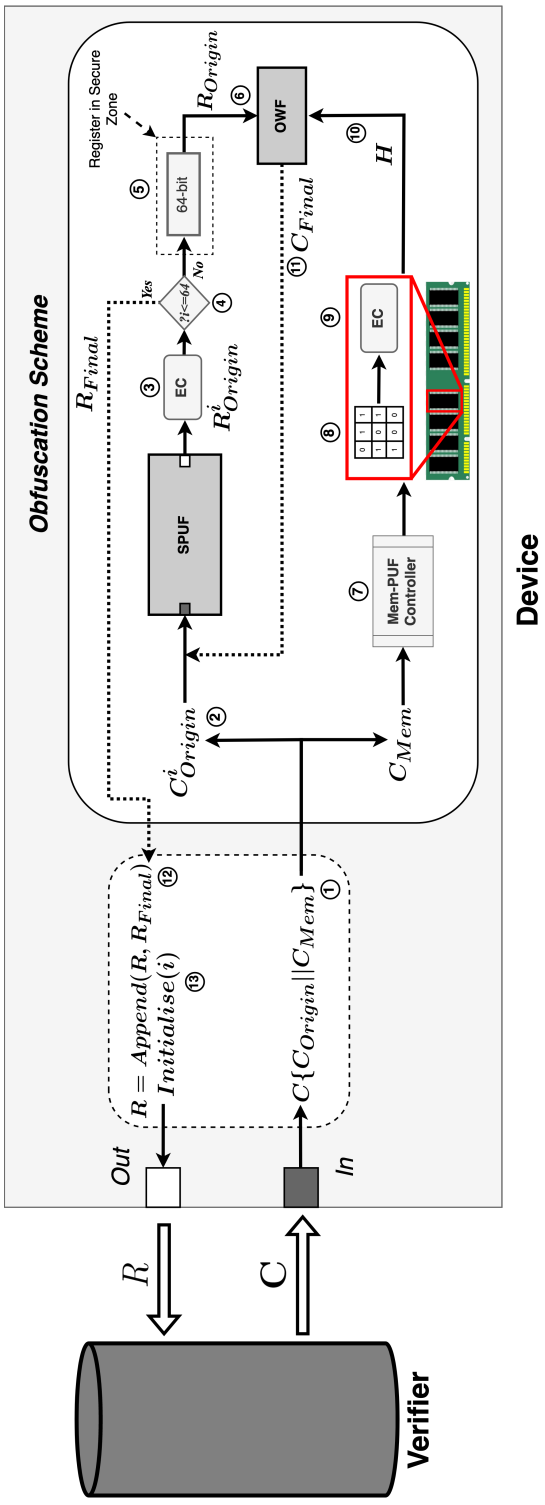

Figure 4.1: *Generic PUF Obfuscation Scheme*

4.4 Experimental Methodology and Discussion

In this section, we first provide details of the setup/methodology used to validate the proposed obfuscation scheme, including details on the PUF datasets and quality of DRAM-PUF measurements.

4.4.1 Strong PUF Datasets

As with most hardware cryptographic primitives, adjusting PUF parameters can enable a designer to tailor the PUF design around the properties most desired for a given security scheme. With Arbiter-based PUFs, one such parameter which can be varied is length of the PUF (referred to as 'stages') which is adjusted through variation of the number of multiplexers used in the delay chain (see Figure 2.2). Increasing the size of the PUF has an exponential effect on increasing or decreasing the available CRP space, in theory enhancing the security of the PUF at the cost of decreased reliability. As shown in [74], while longer APUFs require more resources to model with ML-MA, APUFs of even 128-bits in length can be broken fairly trivially with sufficient CRP training data. While some may argue these render such PUFs redundant, we propose it as a reason to utilise the PUF for its strength in the ability to generate many unique CRPs on-the-fly without any kind of additional processing, while security against ML is offloaded to other mechanisms in the overall scheme. For this reason, we propose the use of smaller APUFs, and thus verify our proposed scheme on 16-stage and 32-stage APUF variants, where the 16-stage APUFs support 2^{16} unique CRPs and the 32-stage APUFs support 2^{32} CRPs. This design choice intrinsically reduces hardware overhead simply due to the smaller PUF size, but also inevitably reduces further error correction overheads as PUF reliability is not worsened with a larger PUF. For a 16-stage PUF, while it may seem only to support a relatively low number of possible unique CRPs (65,536), our scheme resolves this issue through intrinsic support for reconfigurability. As the challenge/response behaviour of the entire scheme is dependent on both the Strong-PUF challenge and the unique matrix, H from the Memory-PUF, the number of possible Strong-PUF CRPs scales directly with the number of unique responses supported by the Memory-PUF. In this way, just 10 unique Memory-PUF responses brings the total supported CRPs of the proposed scheme to 655,360 CRPs, providing an enhancement for scalability. Additionally, this has the effect of improved resistance to ML-MA as when the scheme is reconfigured, an adversary must attempt to train a new ML model to capture the new CRP behaviour of the scheme (discussed further in Section 4.5.5).

We utilised the PyPuf Python framework to generate a set of Strong PUF datasets, through simulating the linear delay models of each PUF tested [89]. To cover a variety of APUF variants, we tested both 16-stage and 32-stage APUF, XOR-APUF and FF-APUF. These PUFs have well-defined mathematical models which describe their behaviour and can therefore be accurately simulated in this context. These simulations may not precisely emulate the behaviour of specific hardware implementations of these PUFs, but they do accurately represent the general behaviour and, crucially, their modelling complexity. As this work aims to evaluate a counter-modelling scheme and the simulated PUFs are an accurate representation of the work needed to perform a modelling attack they are entirely suitable for use here.

4.4.2 Memory PUF Dataset

To generate the H-matrices for our OWF, we opted to use a Latency DRAM-PUF due to the size of available responses, ideal PUF properties, high measurement speed and the ubiquity of DRAM in commodity devices. It should be noted that this is only a test case using a particularly well-suited PUF design. In theory, any weak PUF which can quickly generate large responses could perform the same function. Unlike the strong PUF designs used, DRAM-based PUF behaviour has only been observed experimentally. There is no defined mathematical model by which the physical level behaviour can be simulated. It would be possible to simply assign each simulated cell a probability of failure, but this misses many nuances such as the influence of surrounding cell contents, line variation, sense amplifier variation, etc. Developing and verifying the accuracy of such a model is beyond the scope of this work, therefore we chose to source DRAM PUF data generated experimentally from hardware. The experimental setup for this used the same setup described in [63], targeting Commodity Off-The-Shelf (COTS) DIMM form factor desktop DRAM modules. In each experiment a data pattern was written to memory, then the timing parameter *tRCD* was lowered to 0 clock cycles. Sequential attempts were made to read the data pattern with the results of these attempted read operations forming the PUF response. No error correction or filtering was applied. Data was generated for test patterns 0x00 (all '0'), 0xFF (all '1') and 0x55 (checkerboard pattern). For use in the proposed scheme, each DRAM PUF response was transposed to match the required size of our H-matrices. Error correction was not necessary as only a single execution of the obfuscation scheme was required for each given DRAM-PUF measurement, meaning we can assume the DRAM-PUF measurement used is the final 'golden response'.

4.4.3 Equipment Used

Finally, we performed each of our ML attacks using Python 3.8.12, PyPuf and ScikitLearn on an Intel i9-11980Hk CPU @ 2.60GHz and DDR4 3200MHz 64GB memory. Also, in order to observe the total hardware overhead, we implemented our proposed scheme on a Xilinx Zynq-7000 FPGA device using Xilinx Vivado.

4.4.4 DRAM-PUF Characterisation

As discussed in Section 4.2.1, it is imperative for the security of the OWF that the supplied H-matrices are cryptographically significant, such that they exhibit ideal uniformity on the final output. The strong security properties of DRAM-PUFs have been demonstrated experimentally in previous works [41, 68]. In order to verify that the properties seen in previous works held true for the data generated for use in this work, we configured the OWF with DRAM-PUF-generated H-matrices to test the uniformity of outputs for arbitrarily generated OWF inputs. As discussed in Section 3.2.2 (Chapter 3), the challenge for the DRAM-PUF can be configured by two means: memory location and input pattern, each combination of which (ideally) produces a unique response output pattern. Further, the OWF output may be configured through incrementing or decrementing the OWF w value, therefore we include this parameter to enable an increased challenge space, which enables unique OWF configurations totalling: *unique memory locations * input patterns (3) * w*. We refer to a combination of H-matrix and w value from now on as the C^{OWF}. We configured the OWF with each unique

C^{OWF} and tested the outputs over 10,000, 20,000 and 30,000 (separately) unique samples for uniformity. Practically, this process would occur during the enrollment phase of the scheme, where specific combinations of memory location, input pattern and OWF w value are tested for knowledge in advance of which unique challenges should be used during authentication. We found that the DRAM-Latency PUF exhibits a very strong ability to generate cryptographically significant H-matrices. Across our tests, we found that uniformity was ideal (50 +/- 0.5) in more than 80% of cases, with 81.3% of available unique challenge space (memory locations * w) demonstrating ideal uniformity for challenge pattern 0x00, 88.4% for pattern 0xFF and 100% for pattern 0x55. The observed trend of solid pattern (0x00, 0xFF) inputs resulting in reduced uniformity in some memory regions while mixed pattern inputs produce near-ideal uniformity is consistent with the results reported in [41] [68]. The plots shown in figures 4.2, 4.3 and 4.4 highlight the distributions of uniformity measurement for 0x00 (Pat 0), 0xFF (Pat 1) and 0x55 (Pat M) measurements respectively. The observed results demonstrate $w = 12$ to be the most effective across each H-matrix in ensuring ideal uniformity of the OWF outputs. Generally across the 0x00 and 0xFF configurations, the outputs trend towards the lower bound of uniformity (49.5) than the upper bound (50.5). The 0xFF H-matrices demonstrated extremely ideal uniformity across all w configurations, with even outliers falling well within the ideal upper and lower bounds. Interestingly, using $w = 7$ for both Pat 1 and Pat 0 display abnormally worse uniformity than the other values tested, bringing the uniformity *above* 50.5 for Pat 0 and *below* 49.5 for Pat 1. An intuition for the cause of this observation is that there is a reduced complexity when splitting the message into fewer sets as overall fewer operations will be carried out over the entire message.

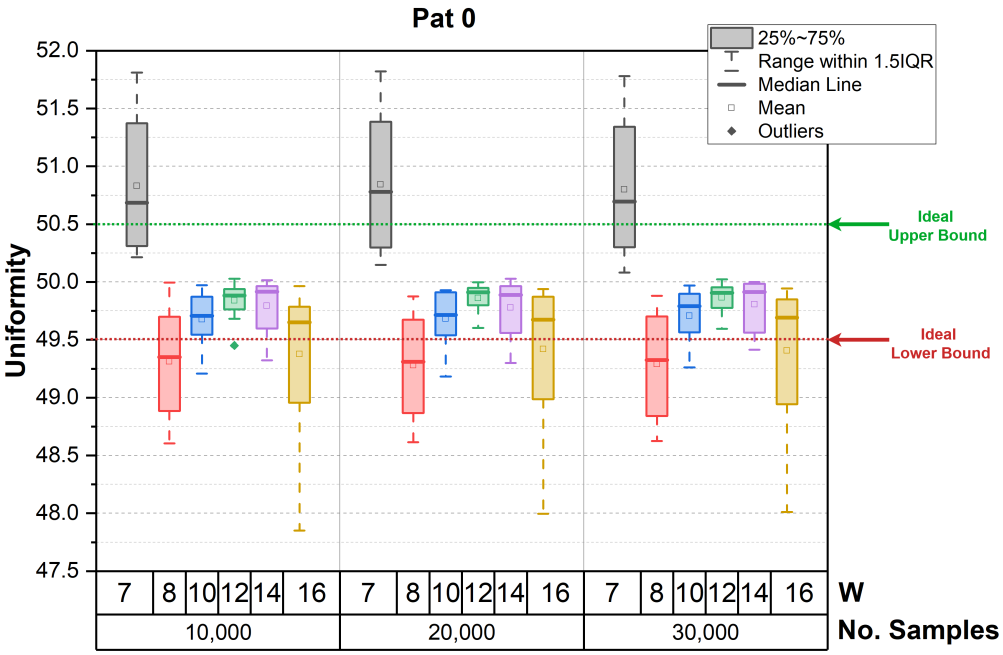

Figure 4.2: *Uniformity results for All Zeros pattern DRAM-PUF H-Matrices*

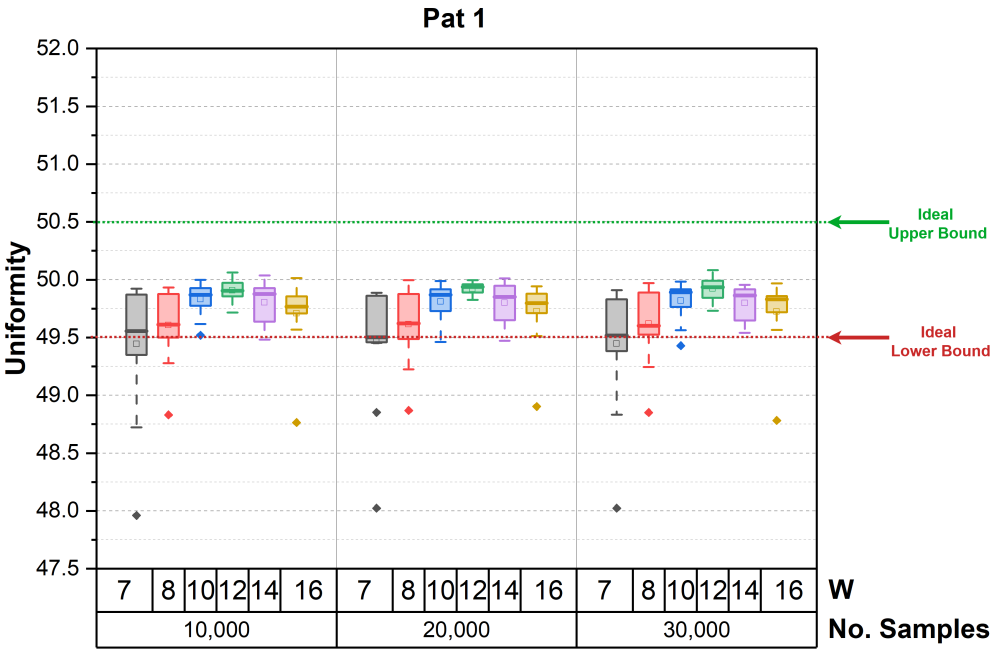

Figure 4.3: *Uniformity results for All Ones Pattern DRAM-PUF H-Matrices*

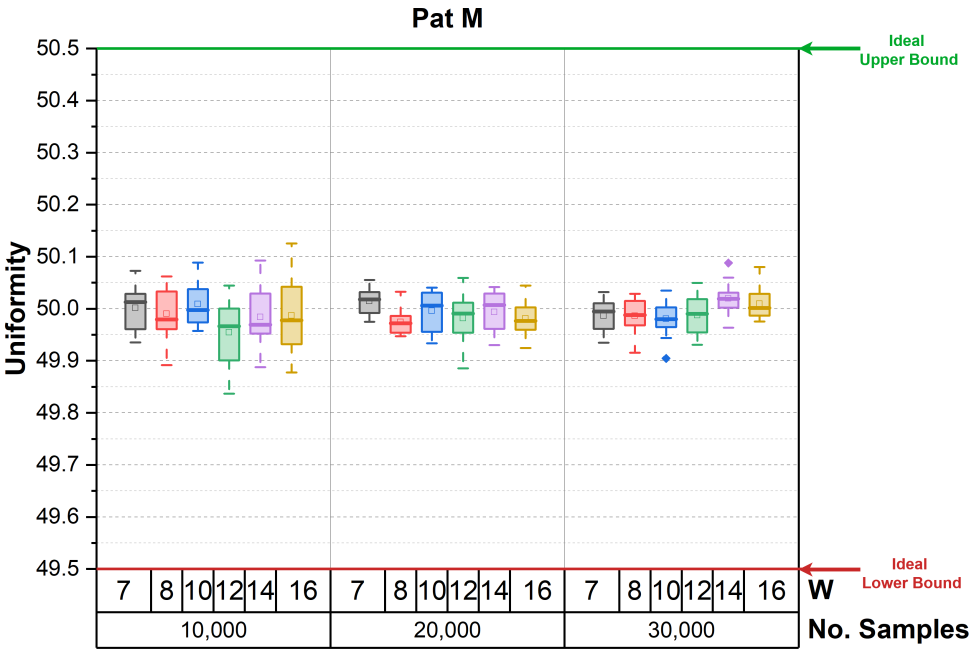

Figure 4.4: *Uniformity results for Checkerboard pattern DRAM-PUF H-Matrices*

4.5 Results and Analysis

In this section, we present the obtained results and discuss our observations.

Table 4.2: *ML-MA using Classic Methods*

(a) Experiment 1: ML-MA without proposed obfuscation scheme (C_{Origin} -> R_{Origin})

PUF Type	Num Stage	Attack	Training/Testing CRPs	Accuracy
APUF	16	LR	5000	0.944
		MLP	5000	0.988
	32	LR	10000	0.977
		MLP	10000	0.987
XOR-APUF	16	LR	50000	0.974
		MLP	50000	0.989
	32	LR	100000	0.944
		MLP	100000	0.984
FF-APUF	16	LR	50000	0.603
		MLP	50000	0.97
	32	LR	100000	0.645
		MLP	100000	0.981

(b) Experiment 2: ML-MA with proposed obfuscation scheme (C_{Origin} -> R_{Final})

PUF Type	Num Stage	Attack	Training/Testing CRPs	Accuracy
APUF	16	LR	720000	0.53
		MLP	720000	0.531
	32	LR	360000	0.496
		MLP	360000	0.477
XOR-APUF	16	LR	720000	0.525
		MLP	720000	0.529
	32	LR	360000	0.509
		MLP	360000	0.517
FF-APUF	16	LR	720000	0.484
		MLP	720000	0.521
	32	LR	360000	0.498
		MLP	360000	0.478

Table 4.3: *Custom ML-MA Results*

Model	PUF	Num. Stages	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
Logistic Regression	APUF	16	0.0630	0.5021	0.0629	0.0630	0.0629	0.0005	0.0005
		32	0.2485	0.4979	0.2483	0.2483	0.2482	-0.0023	-0.0023
	XOR-APUF	16	0.0640	0.4991	0.0638	0.0638	0.0638	0.0014	0.0014
		32	0.2485	0.4979	0.2483	0.2483	0.2482	-0.0023	-0.0023
	FF-APUF	16	0.0621	0.5002	0.0621	0.0621	0.0620	-0.0005	-0.0005
		32	0.2524	0.5014	0.2524	0.2524	0.2524	0.0031	0.0031
Linear Discriminant Analysis	APUF	16	0.0629	0.5021	0.0628	0.0629	0.0628	0.0003	0.0003
		32	0.2484	0.4979	0.2481	0.2481	0.2480	-0.0025	-0.0025
	XOR-APUF	16	0.0638	0.4992	0.0636	0.0636	0.0635	0.0012	0.0012
		32	0.2484	0.4979	0.2481	0.2481	0.2480	-0.0025	-0.0025
	FF-APUF	16	0.0556	0.4505	0.0556	0.0555	0.0555	-0.0007	-0.0007
		32	0.2522	0.5014	0.2522	0.2522	0.2522	0.0029	0.0029
Quadratic Discriminant Analysis	APUF	16	0.0627	0.5000	0.0626	0.0624	0.0513	0.0000	0.0001
		32	0.2480	0.4970	0.2477	0.2478	0.2476	-0.0030	-0.0030
	XOR-APUF	16	0.0622	0.4987	0.0617	0.0617	0.0612	-0.0008	-0.0008
		32	0.2480	0.4970	0.2477	0.2478	0.2476	-0.0030	-0.0030
	FF-APUF	16	0.0620	0.4997	0.0620	0.0615	0.0452	-0.0006	-0.0005
		32	0.2506	0.5005	0.2506	0.2506	0.2506	0.0008	0.0008
Ridge Classifier	APUF	16	0.0627	0.0000	0.0627	0.0627	0.0625	0.0002	0.0002
		32	0.2484	0.0000	0.2481	0.2481	0.2480	-0.0025	-0.0025
	XOR-APUF	16	0.0637	0.0000	0.0635	0.0635	0.0633	0.0011	0.0011
		32	0.2484	0.0000	0.2481	0.2481	0.2480	-0.0025	-0.0025
	FF-APUF	16	0.0619	0.0000	0.0618	0.0618	0.0617	-0.0007	-0.0007
		32	0.2522	0.0000	0.2522	0.2522	0.2522	0.0029	0.0029
Naïve Bayes	APUF	16	0.0629	0.5023	0.0628	0.0629	0.0628	0.0004	0.0004
		32	0.2494	0.4982	0.2492	0.2492	0.2491	-0.0011	-0.0011
	XOR-APUF	16	0.0634	0.4994	0.0632	0.0633	0.0632	0.0008	0.0008
		32	0.2494	0.4982	0.2492	0.2492	0.2491	-0.0011	-0.0011
	FF-APUF	16	0.0619	0.5004	0.0619	0.0619	0.0618	-0.0006	-0.0006
		32	0.2527	0.5013	0.2527	0.2528	0.2527	0.0037	0.0037
Decision Tree	APUF	16	0.0626	0.5001	0.0626	0.0626	0.0626	0.0001	0.0001
		32	0.2482	0.4988	0.2482	0.2482	0.2482	-0.0024	-0.0024
	XOR-APUF	16	0.0632	0.5003	0.0631	0.0631	0.0631	0.0007	0.0007
		32	0.2482	0.4988	0.2482	0.2482	0.2482	-0.0024	-0.0024
	FF-APUF	16	0.0639	0.5007	0.0638	0.0639	0.0639	0.0014	0.0014
		32	0.2495	0.4997	0.2495	0.2495	0.2495	-0.0007	-0.0007
Random Forrest	APUF	16	0.0623	0.4997	0.0622	0.0622	0.0617	-0.0003	-0.0003
		32	0.2509	0.4983	0.2509	0.2510	0.2509	0.0012	0.0012
	XOR-APUF	16	0.0626	0.4998	0.0627	0.0626	0.0624	0.0002	0.0002
		32	0.2509	0.4983	0.2509	0.2510	0.2509	0.0012	0.0012
	FF-APUF	16	0.0630	0.5013	0.0630	0.0629	0.0624	0.0005	0.0005
		32	0.2500	0.5002	0.2500	0.2499	0.2495	-0.0001	-0.0001
Extra Trees Classifier	APUF	16	0.0615	0.4988	0.0614	0.0614	0.0609	-0.0011	-0.0011
		32	0.2499	0.4990	0.2499	0.2499	0.2499	-0.0002	-0.0002
	XOR-APUF	16	0.0624	0.4998	0.0625	0.0625	0.0623	0.0000	0.0000
		32	0.2499	0.4990	0.2499	0.2499	0.2499	-0.0002	-0.0002
	FF-APUF	16	0.0625	0.4988	0.0624	0.0623	0.0617	-0.0001	-0.0001
		32	0.2510	0.5021	0.2509	0.2510	0.2505	0.0012	0.0012

(a) *Custom supervised ML-MA results*

Model	PUF	Num. Stages	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
DQN	APUF	16	0.0637	0.0000	0.0636	0.0635	0.0634	0.0012	0.0012
		32	0.2523	0.5000	0.2522	0.2523	0.2523	0.0030	0.0030
	XOR-APUF	16	0.0639	0.0000	0.0635	0.0634	0.0633	0.0012	0.0012
		32	0.2532	0.5021	0.2532	0.2533	0.2532	0.0042	0.0042
	FF-APUF	16	0.0636	0.5012	0.0635	0.0634	0.0632	0.0011	0.0011
		32	0.2540	0.5026	0.2539	0.2539	0.2538	0.0052	0.0052
HDBSCAN	APUF	16	0.0635	0.4995	0.0635	0.0634	0.0634	0.0010	0.0010
		32	0.2517	0.5002	0.2511	0.2511	0.2505	0.0015	0.0015
	XOR-APUF	16	0.0639	0.5012	0.0637	0.0637	0.0637	0.0014	0.0014
		32	0.2511	0.5006	0.2509	0.2510	0.2509	0.0012	0.0012
	FF-APUF	16	0.0628	0.5006	0.0627	0.0629	0.0622	0.0003	0.0003
		32	0.2531	0.0000	0.2531	0.2531	0.2531	0.0041	0.0041
DBSCAN	APUF	16	0.0632	0.4996	0.0632	0.0631	0.0631	0.0007	0.0007
		32	0.2510	0.5001	0.2509	0.2510	0.2509	0.0012	0.0012
	XOR-APUF	16	0.0637	0.4993	0.0630	0.0629	0.0616	0.0005	0.0005
		32	0.2494	0.4982	0.2486	0.2484	0.2476	-0.0019	-0.0019
	FF-APUF	16	0.0621	0.5003	0.0621	0.0621	0.0620	-0.0004	-0.0004
		32	0.2531	0.5026	0.2530	0.2531	0.2530	0.0041	0.0041
BIRCH	APUF	16	0.0632	0.4993	0.0632	0.0631	0.0631	0.0007	0.0007
		32	0.2491	0.4990	0.2488	0.2489	0.2488	-0.0016	-0.0016
	XOR-APUF	16	0.0636	0.5011	0.0634	0.0633	0.0633	0.0010	0.0010
		32	0.2490	0.5012	0.2488	0.2489	0.2489	-0.0016	-0.0016
	FF-APUF	16	0.0620	0.0000	0.0619	0.0619	0.0618	-0.0006	-0.0006
		32	0.2530	0.5025	0.2530	0.2530	0.2529	0.0040	0.0040
K-Means++	APUF	16	0.0631	0.5008	0.0630	0.0633	0.0627	0.0006	0.0006
		32	0.2491	0.0000	0.2488	0.2489	0.2488	-0.0016	-0.0016
	XOR-APUF	16	0.0636	0.5013	0.0634	0.0634	0.0634	0.0010	0.0010
		32	0.2483	0.4985	0.2479	0.2480	0.2478	-0.0029	-0.0029
	FF-APUF	16	0.0618	0.4996	0.0618	0.0619	0.0618	-0.0008	-0.0008
		32	0.2524	0.5023	0.2524	0.2524	0.2524	0.0032	0.0032
K-Means	APUF	16	0.0625	0.5015	0.0622	0.0622	0.0617	-0.0003	-0.0003
		32	0.2490	0.4987	0.2487	0.2488	0.2487	-0.0017	-0.0017
	XOR-APUF	16	0.0635	0.5005	0.0635	0.0635	0.0635	0.0010	0.0010
		32	0.2474	0.0000	0.2470	0.2471	0.2469	-0.0040	-0.0040
	FF-APUF	16	0.0617	0.5000	0.0617	0.0617	0.0616	-0.0009	-0.0009
		32	0.2504	0.5016	0.2503	0.2506	0.2499	0.0004	0.0005
K-Medoids	APUF	16	0.0614	0.4986	0.0613	0.0611	0.0607	-0.0012	-0.0012
		32	0.2486	0.4987	0.2484	0.2485	0.2484	-0.0022	-0.0022
	XOR-APUF	16	0.0630	0.5008	0.0630	0.0630	0.0627	0.0005	0.0005
		32	0.2474	0.4978	0.2470	0.2471	0.2470	-0.0040	-0.0040
	FF-APUF	16	0.0617	0.4996	0.0617	0.0617	0.0611	-0.0009	-0.0009
		32	0.2502	0.5009	0.2502	0.2502	0.2497	0.0002	0.0002
Gaussian	APUF	16	0.0612	0.4993	0.0612	0.0612	0.0611	-0.0014	-0.0014
		32	0.2479	0.4985	0.2478	0.2479	0.2478	-0.0029	-0.0029
	XOR-APUF	16	0.0625	0.4997	0.0625	0.0626	0.0623	0.0000	0.0000
		32	0.2473	0.4976	0.2469	0.2470	0.2469	-0.0041	-0.0041
	FF-APUF	16	0.0617	0.5002	0.0617	0.0616	0.0616	-0.0009	-0.0009
		32	0.2488	0.4992	0.2488	0.2488	0.2487	-0.0017	-0.0017

(b) *Custom unsupervised ML-MA results*

4.5.1 Resilience to ML-MA

Here, we first report the findings of classic PUF ML-MA methods on the Strong-PUFs both with and without the proposed scheme. After, we report the findings of our custom ML attack on the scheme. In order to test a variety of PUF types, we performed all tests on both 16-stage and 32-stage variants of an APUF, XOR-APUF and FF-APUF (described in Sections 2.4.3 2.4.4 and 2.4.5 respectively).

4.5.1.1 Classical ML-MA

In order to benchmark our proposed scheme against the classic ML-MA methods on PUFs, we first performed both the Logistic Regression (LR) attack proposed in [74] (Section 2.5.1) and the Multi-Layer Perceptron (MLP) attack proposed in [1] (Section 2.5.2) on both the PUFs with and without the proposed scheme. For a fair comparison, we conducted two experiments. In this first experiment, we perform both the LR and MLP attacks on the Strong-PUFs without the obfuscation scheme, such that we assume an adversary has access to the Strong-PUF challenges and responses, C_{Origin} and R_{Origin}, and aims to predict direct new R_{Origin} responses from the same PUF. In the second experiment, we consider the threat model

described in 4.2.3, where an attacker gains access to the initial challenges and final responses of the obfuscation scheme, C_{Origin} and R_{Final}, and aims to predict new final response bits. We trained each model for the 16-stage APUF on 5000 CRPs and 10,000 CRPs for the 32-stage APUF. Due to the increased complexity of the XOR-APUF and FF-APUF, for consistency we trained each model on 10x the number of CRPs as the APUF variants, though it is possible to successfully model these PUFs with fewer CRPs as demonstrated in [74] and [1], especially considering they have few stages. Finally, the training CRPs were split into sets of 99% for training and 1% for validation. We utilised this large 99% training set due to the very large set of samples available, allowing maximal attacker knowledge, supported with the fact that statistical significance in predictive capability can be captured with 3,200 and 7,200 CRPs (1% of total set). Table 4.2(a) provides the details and prediction accuracies of the classical ML-MA attacks. Our benchmark showed expected results, where prediction accuracies for each PUF remained above 94% for each attack, except the FF-APUF for the LR attack with around 60% accuracy due to the enhanced resilience against linear classifiers. When faced with the MLP attack, the prediction accuracy matched that of the other PUFs at above 97%. However, when the obfuscation scheme is implemented, the modelling results show a significantly reduced performance. Each PUF was attacked using the maximum available CRPs that were generated in the dataset, 720,000 CRPs for the 16-stage APUF (chosen arbitrarily based on practical storage limitations of 1024-bit challenges) and 320,000 CRPs for the 32-bit APUF, determined as 720,000 divided by two due to twice as many challenge bits being required by the 32-stage APUF than the 16-stage APUF. As larger challenges are required per final response bit, more possible CRPs are available for an attacker to train the model. Table 4.2(b) shows the results of the classical ML attacks when integrating the proposed scheme. Each attack performed for each PUF showed an average prediction average of 50%. Even when using the most vulnerable 16-bit APUF, the maximum prediction accuracy was extremely low at 53.1%, which is almost the equivalent of a random coin flip. This result demonstrates the potent ability of the scheme to obscure any linearity between the original challenge and final response, such that an attacker is provided with no advantage when attempting an ML attack.

4.5.1.2 Custom ML-MA

As the proposed scheme deviates from the simple additive linear delay model of the Strong-PUFs on their own and the number of final output bits available to an attacker, it is also necessary to test a wider variety of classifiers against our proposed scheme. We first benchmark the performance of a set of supervised classification algorithms consisting of Decision Tree, Random Forest, Extra Trees, Logistic Regression, Quadratic Discriminant Analysis, Naive Bayes, Ridge, Light Gradient Boosting Machine, and Linear Discriminant Analysis, the results of which are depicted in Table 4.3a. We optimise the performance of each ML classifier by performing a random grid search using a predefined internal grid used within the Pycaret Python framework , the ranges of which are provided in Table 4.4. Each model is evaluated using raw accuracy, AUC, recall, precision, F1, Kappa and MCC [10, 32]. For brevity, the details of each metric have been provided in Appendix 1 for the interested reader.

We also benchmark the performance of our proposed scheme against a set of unsupervised algorithms, namely Deep Q-Network, HDBSCAN, DBSCAN, BIRCH, K-Means++, K-Means, K-Medoids and Gaussian. Unsupervised classifiers, where obscure connections in

an unlabeled dataset are discovered by grouping data into clusters or by association. Most importantly, we evaluate our proposed scheme against a reinforcement learning (RL) based attacker using Deep Q-Networks (DQN). DQN is an approximation-based RL where an agent interacts with the environment by sensing its state and learns to take action to maximise long-term reward. As the agent takes action, it needs to maintain a balance between exploration and exploitation by performing a variety of actions using trial and error in an uncertain environment to favour the actions that yield the maximum reward in the future. This type of exploratory attack is well suited for learning highly non-linear relationships/correlations within a complex feature space, such as is required for modelling PUF obfuscation schemes, similar to the CMA-ES attack performed in [6]. The results of the unsupervised classifiers are shown in Table 4.3b

In order to ensure the generalisation and verify results, we perform 10-fold-cross validations on each experiment experiment. The challenges (training features) consisted of 1024 and 2048 features (for each bit) for the 16-stage and 32-stage PUF, respectively. As the scheme outputs 4 bits for the 16-stage Strong PUFs and 2 bits for the 32-stage Strong PUFs, the classification labels were 4 and 2 bits, respectively (15 and 4 labels).

Our results showed each classifier faced extreme difficulty in detecting a relationship between the challenge and response data for both the supervised and unsupervised classifiers, with the accuracy of each classifier for each PUF type not exceeding 6.3% for the 16-bit PUFs and 20% for the 32-bit PUFs. Overall, the unsupervised classifiers performed slightly better overall than the supervised methods, which is intuitive given the suitability of many unsupervised classification methods to non-linear predictive tasks such as those required for PUF obfuscation modelling.

Table 4.4: *Hyperparameter tuning ranges for custom supervised classifiers*

Parameter	Description	Grid Search Ranges
Logistic Regression		
C	Type of regularisation	0.01, 0.1, 1, 10, 100
Max Iterations	Number of iterations	100, 200, 300, 500, 1000
Linear Discriminant Analysis		
Solver	Algorithm used	svd, lsqr, eigen
Quadratic Discriminant Analysis		
Reg_param	Regularisation parameter	0.0, 1.0
Ridge Classifier		
Alpha	Regularisation strength	0.1, 1, 10, 100, 1000
Solver	Algorithm used	auto, svd, cholesky, lsqr, sparse_cg, sag, saga
Fit_intercept	Whether to calculate the intercept for the model	True, False
Normalise	Whether to normalise features before training	True, False
Naive Bayes		
Var_Smoothing	Controls portion of the largest variance of all features for calculation stability	1e-9, 1e-8, 1e-7, 1e-6, 1e-5
Decision Tree		
Criterion	Function used to measure the quality of split	Entropy, Gini
Max Depth	Maximum tree depth	5, 10, 20, 50, 100
Leaf Min Samples	Min number of samples required at leaf node.	1, 2, 5, 8, 16, 32
Random Forest		
Estimators	Number of trees	10, 50, 100, 200
Max Features	Set as function of number of features	sqrt, log2
Max Depth	Maximum tree depth	None, 3, 5, 10, 20
Min Samples Split	Minimum samples needed to split a node	2, 5, 10
Min Samples Leaf	Minimum samples needed to split a leaf	1, 2, 4
Extra Trees Classifier		
Estimators	Number of trees	100-500

4.5.2 Security Analysis

In this section, we will briefly discuss various potential threats to the proposed system and the implications of those threats. It should be noted that the claims being made in this chapter relate to security against remote ML-MA only, as defined in the threat model in Section 4.2.3. Resistance to attacks requiring physical access is left for future work. Nonetheless, we will discuss some such attacks in order to provide the reader with a clear picture of where this countermeasure fits into the overall threat landscape.

4.5.2.1 ML-MA Remote Attacks

As shown in Section 4.5.1, the countermeasures in the scheme render modeling the system as a whole infeasible if only the overall inputs and outputs are known and not the internal states. These inputs and outputs are the only items of information transmitted over open channels; therefore, this strongly impedes remote modeling attacks.

4.5.2.2 Replay Attacks

As the challenges and responses are being transmitted openly, and it is explicitly assumed the adversary can listen in on this channel, there is a requirement at the protocol level to invalidate CRPs once they have been used. Otherwise, there is a risk that if a previously used challenge is issued for which the adversary recorded the response, they would be able to provide the correct response despite no knowledge of the PUF. There is a similar need to prevent an adversary from issuing rapid false challenges to gather the full CRP set through rate-limiting challenges at the device side, removing CRPs if a response is received unexpectedly or both.

4.5.2.3 Cold Boot Attacks

As the scheme uses a memory PUF to generate matrices for the OWF, attacks which aim to capture data from volatile memories such as cold boot attacks have the potential to compromise the matrices. How dangerous is this in practice? Consider a scenario where the adversary knows the n column matrix, H, the OWF padding method, the OWF block number, w, and the output of the OWF, R_{Final}. They want to learn the OWF input (i.e. the raw PUF output). To reverse the OWF with this information requires them to guess which columns are XORed, which will require n^w trials. e.g. using the value of $n = 1024$ in our example scheme and $w = 16$ would require in the order of 2^{160} trials. Thus, H being known to the adversary *does* reduce the complexity of OWF reversal, but not enough to meaningfully compromise the system.

4.5.2.4 Memory Snooping Attacks

Attacks which allow snooping of system memory or manipulation of the memory controller (e.g. malicious software) could conceivably leak the matrix set, which is generated from memory and resides temporarily in it. The same analysis as the previous point applies here, where knowing the matrices isn't enough in itself to compromise the system. The only other aspects which ever reside in system memory are the challenges and final responses which we assume the adversary can acquire over the transmission channel anyway.

4.5.2.5 Timing Side-Channel Attacks

While the OWF itself is time-invariant, this is not necessarily true for the PUF, or for the PUF error correction. In fact, *because* the OWF is fixed time the timing of the whole scheme reveals information about the timing of the PUF+ECC block. This information could conceivably be used to assist model construction and make the scheme easier to compromise, although the exact efficacy of such an attack is uncertain. Ideally, the PUF block and ECC should be made time-invariant, which removes this channel of information leakage. This is especially relevant to this work as timing can be analysed remotely to a degree, unlike other side channels, which require physical access.

4.5.2.6 EM Side Channel Attacks

If the adversary has physical access to the target device, there may be attacks which could be employed using EM side-channel analysis. As with the other side-channel attacks, there are two separate risks. First, direct measurement of the PUF as it operates. Second, the measurement of OWF internal states breaks the security properties of the function. The actual degree of leakage through EM emission and how to mask it if necessary are outside the scope of this work, but it should be noted as a factor for consideration in future work.

4.5.2.7 Voltage Side Channel Attacks

As with the EM emissions, there may be some information which can be derived about the internal states through monitoring of power consumption during operation. Again, exploration of this, and how to level out power use in the scheme if necessary, is outside the scope of this work but may be explored in future works.

4.5.2.8 Fault Injection Attacks

The points above assume correct operation, but there may be additional risks if the adversary intentionally causes faults in parts of the system. For example, it may be possible to partially bypass the complexity of the attack mentioned in Section 4.5.2.3 by selectively faulting one column of H at a time while repeatedly issuing the same challenge. Or issuing falsified challenges and selectively faulting the responses such that the protocol does not invalidate the collected CRP. Such attacks unequivocally require prolonged physical access and are outside the scope of this work, but they provide interesting possibilities for future work and are worth noting.

4.5.3 Hardware Overhead

Strengths and weaknesses based on certain included features tend to map linearly to required hardware overhead, as demonstrated by similar obfuscation schemes. The hardware overhead of our scheme includes the Strong-PUF utilised, PUF error correction, 64-bit buffer and the OWF. Different types and/or sizes of Strong PUFs will incur varying hardware overhead; however, in our scheme, some overhead is mitigated through the use of smaller-length PUFs. Comparable schemes such as [95] employ 64 to 128-bit Arbiter PUFs, whereas our scheme demonstrated protection for even 32-stage PUFs. Due to the suitability of our scheme

with APUFs, we benchmark our scheme using the hardware overhead of 16-bit and 32-bit APUFs. As we propose a generic obfuscation scheme, the required ECC will vary as different Strong-PUFs are used for a token generation as different PUF types incur different reliability properties [37] which can have varying effects on the overall hardware overhead of the scheme. PUFs tend to exhibit a higher error rate as the size and complexity of the circuitry grow; for example, the more individual APUFs used to construct an XOR-APUF, the higher the error rate tends to be. The APUF implementation in [36] can achieve a Bit Error Rate (BER) of as low as 10^{-9}, meaning lower cost ECC schemes that rely on high PUF reliability such as provided by Hiller et al. in [38] may be implemented. Therefore, given our proposed benchmark, it is possible to utilise a highly reliable APUF and thus employ far less resource-consuming ECC methods. For a more comprehensive ECC implementation, we separately performed majority voting [15] and the Golay (23,12,7) code [48] for both 16-stage and 32-stage variants of an APUF. We performed a synthesis on the programmable logic of a Xilinx Zynq-7000 FPGA to determine the hardware overhead. Figure 4.5 shows the chip layout for the synthesised generic PUF obfuscation scheme. The total resource requirements of our scheme are listed in Table 4.5.

Table 4.5: *Comparison of the Hardware and Power Overhead of the Proposed Scheme Against the State-of-the-art*

Scheme	**LUTs**	**DFFs**	**Power (*W*)**
Controlled PUF [28]	1830	3020	∼
PUF-FSM [24]	960	1500	∼
Set-based [95]	395	1400	∼
CT-PUF [96]	741	486	0.107
Proposed Scheme			
APUF: † **ECC:** ‡‡	306	298	0.177
APUF: † **ECC:** ††	291	330	0.178
APUF: ‡ **ECC:** ‡‡	341	312	0.180
APUF: ‡ **ECC:** ††	327	345	0.179

† 16-Bit ‡ 32-Bit †† Golay Code ‡‡ Majority Voting ∼ Data not available

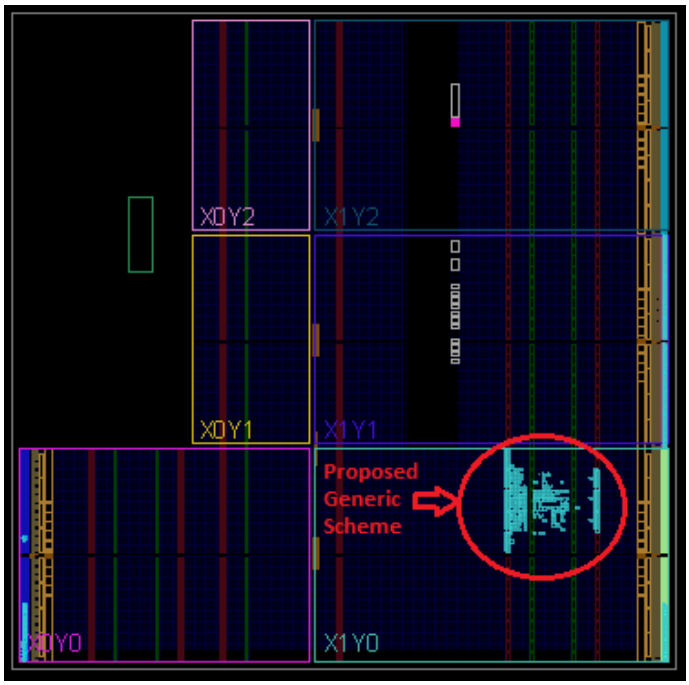

Figure 4.5: *Layout of the Proposed Generic PUF Obfuscation Scheme (16-bit APUF and Majority Vote ECC) on a Xilinx Zynq-7000 FPGA Device*

4.5.4 Power Consumption

As the memory PUF operates on existing components, namely the ARM Cortex A9 processor cores of the Zynq and an external DDR3 DRAM chip, power consumption for it is a function of how much additional load the required instructions add to the Processor System (PS). Each matrix generation requires a number of memory accesses that scales proportionally with the number of rounds of majority vote ECC. For matrices of the size used in our experiments, this requires 2048 writes and $2048m$ reads to the DDR memory, where m is the number of rounds. This uses negligible power even for fairly large m (e.g. at $m = 100$ memory access load is increased by 0.034%, with a power consumption of $< 1mW$). Error correction also requires $(258n) \times m$ processor instructions, where n is the bit length of the matrix. For the example matrices and 10 round ECC, this equates to 26% of maximum processor load for one core, which consumes $0.065W$. For the FPGA implementation, we utilised the Xilinx power estimator to determine the power consumption of the part of the scheme which operates on the programmable logic. We found the total power consumption to be very low, using only $0.112\,W$ at minimum (16-bit APUF and Majority Vote ECC) and $0.115\,W$ at maximum (32-bit APUF and Majority Vote ECC). From this, we can simply calculate the total power consumption of one full cycle of the scheme by adding the DRAM-PUF power consumption ($0.065\,W$) to the FPGA power consumption, which gives a total (maximum) of $0.180\,W$. Table 4.5 shows the combined total power cost for different configurations of the overall proposed scheme (FPGA logic and DRAM-PUF).

4.5.5 Implications of Varying Hardware

It should be noted that in addition to the costs listed, there is a variable degree of (D)RAM usage. The footprint of this depends on the memory configuration of the system. When using the Latency DRAM-PUF as in our experiments, an amount of memory is used during matrix generation at least ten times the size of the desired matrix. This is due to the fact that when using the Latency DRAM-PUF, the actual stored data is unaffected by the PUF process; rather, when the memory is read, the controller misreads what is actually stored, and that error pattern is the PUF response. Therefore, when using majority vote error correction, you need an area the size of H to perform the PUF operation on, another equally sized area to store the temporary result and at least one byte per bit of that for per-bit majority vote counting. This is fine so long as there is sufficient DRAM free to allow a region of that size to be available whenever the PUF needs to be queried. If, however, there is no DRAM available or this overhead is not practical, BRAMs on the FPGA logic can be utilised to store matrices instead, at the cost of increasing the overall FPGA hardware footprint of the scheme. Other Memory-PUFs that cause changes directly in the stored data, such as Start-up SRAM-PUF (Section 2.4.6) and Retention DRAM-PUF, can also be used and would have a smaller footprint as the matrix is generated directly in the region of memory being used as the PUF, though there may be some overhead required for error correction. However, these PUFs are not as easy to use in runtime due to their query processes being destructive to data held in the same memory. Given these factors, a trade-off is available to system designers based on which features are most desirable for the given application:

Latency DRAM-PUF:

+ Very fast measurement speed
+ Large space available for many supported unique responses
+ Non-destructive to memory contents
− Additional memory space required to store the response

Retention DRAM-PUF:

+ No additional memory is required to store the response
+ Large space available for many supported unique responses
− Very slow measurement speed
− Destructive to memory contents

Start-up SRAM-PUF:

+ Fast measurement speed
+ No additional memory space is required to store the response
− Lower density, therefore fewer unique responses supported
− Requires power cycle to query
− Destructive to memory contents

4.6 Summary

Physical Unclonable Functions (PUFs) offer a promising solution for the lightweight authentication of IoT devices as they provide unique fingerprints for the underlying devices through their challenge-response pairs. However, PUFs have been shown to be vulnerable to Machine-Learning Modelling-Attacks (ML-MA). In this chapter, we proposed a novel obfuscation scheme for preventing ML-MA on Strong PUFs by exploiting DRAM-PUF and a One-Way Function to obfuscate PUF challenges and responses. We demonstrated our scheme has a significant effect on reducing the accuracy of ML-MA to almost the same probability as a random coin flip per bit when tested against various established attacks from the literature and against our own additionally tested classifiers. Our experimental results also show our scheme has the potential to be very cost-effective with regards to hardware overhead on FPGA-enabled devices. While we have considered an initial hardware-level solution for integrating DRAM-PUFs for authentication, it remains an important

Data Availability

For reproducibility of this work and engagement with the wider research community, all source code (HDL designs and ML attacks) can be found at [65]: `https://doi.org/10.15131/shef.data.27095215.v1`. Additionally, the data for our experiments in this chapter have been made available at [64]: `https://doi.org/10.15131/shef.data.26977279.v1`.

Chapter 5

A Privacy-Preserving Protocol Level Approach to Prevent Machine Learning Modelling Attacks on PUFs in the Presence of Semi-Honest Verifiers

Effective protocol design is essential for ensuring the security of interconnected devices, where adversaries are known to operate on publicly accessible networks. While this type of eavesdropping attacker is commonly considered, more knowledgeable attackers such as insiders are less often targeted as potential adversaries for protocol designs. In this Chapter we propose a protocol-level approach for integrating DRAM-PUF based obfuscation hardware design to prevent ML-MA not only to classical eavesdropping adversaries, but also to a novel 'semi-honest' verifier threat. We provide an amended version of the hardware obfuscation scheme proposed in the previous chapter and tightly integrate it into the functioning of the protocol. Our security analysis demonstrates the effectiveness of the proposed scheme for preventing ML-MA at both the hardware and protocol level against a highly knowledgeable adversary.

5.1 Introduction and Related Work

Secure hardware design is an integral component for ensuring effective RoT in any computerised system. Once hardware-level approaches are successfully considered, it becomes imperative to design communication methodologies which synchronise with specific hardware design in order to maximise the effectiveness when applied to networked systems. With PUFs in mind, it remains vital for parties (and intruders) to be unable to gain sufficient information from communication processes as to gain advantage in compromising a given PUF. As previously introduced, PUFs are often vulnerable to ML-MA, which can be supplemented by additional knowledge learned by adversaries who may gain information from eavesdropping naïve protocols.

Multiple hardware and protocol-level techniques have been proposed that attempt to

mitigate the ML-based attack surface of PUFs for resource-constrained security systems. In 2015, Aysu et al. proposed an end-to-end protocol to enable privacy-preserving PUF-based authentication [4]. However, Gope et al. noted in their work that the protocol was not scalable and could not ensure the untraceability property [30]. Furthermore, this was an early example of PUF protocol which did not consider ML-MA capable adversaries. Later, Yu et al. proposed a lockdown protocol specifically designed to reduce the possible number of CRPs accessible to adversaries by permanently fusing closed the input/output interface of the PUF, preventing access to sufficient data to perform ML-MA [93]. Restricting CRP access to adversaries comes at the cost of restricting CRP access to genuine users, causing significant issues with the protocol's scalability. In 2021, Gu et al. proposed a deception protocol whereby a fake PUF is utilised to increase the difficulty for an attacker in probing and modelling the genuine PUF [33]. While this scheme successfully considers mechanisms at the hardware and software level to prevent ML-MA, Non-Volatile Memory (NVM) is required to store secret data, which, if exposed, can enable ML-MA. Gope et al. proposed a protocol-level approach to prevent ML-MA using a One-Time PUF (OTP) concept, whereby PUF reconfiguration is exploited to allow scalability through many CRPs being supported without statistical significance being shared across each small set of CRPs [31]. While issues surrounding the scalability of PUF protocols are successfully considered, the requirement for NVM (as in [33]) means a weaker adversary model must be assumed. Ebrahimabadi et al. recently proposed an authentication protocol whereby PUF challenges are split across multiple messages in order to restrain adversaries from collecting entire challenges in their CRP database to prevent ML-MA [21]. However, this scheme requires a third party in the protocol as a helper node to enable the challenge-splitting mechanism, which is restrictive for many IoT use cases and increases the attack surface for denial-of-service and desynchronisation attacks.

5.1.1 Problem Statement and Motivation

Traditionally, PUF-based protocols are designed for the resource-restricted end node devices to interact with a fully trusted server that has significant storage and computational capabilities and is assumed to be inaccessible to an adversary. This assumption enables the classical CRP authentication scheme, where a server collects a large subset of all possible PUF CRPs during a secure enrollment phase, and then stores a look-up table to query the PUF and check for correct responses during normal authentication. The weakness of this assumption is highlighted by Chatterjee et al. [13], where also a PUF authentication and key exchange protocol is proposed, which does not require a CRP database to be stored on the verifier. By relieving the requirement to maintain a CRP database, the overall scheme is strengthened in the case that the verifying server is compromised. This work, however, requires a third, independent and fully trusted security association provider, which increases the complexity of the overall scheme. With IoT, there are many scenarios where the initial provider of an end device may not match that of a verifier at some later stage of the device's life cycle. In many IoT applications, devices are installed and run at the consumer's/user's end, where a consumer/user may attempt to manipulate the settings of a specific device, such as a smart meter, in the case of smart grid infrastructure. Here, smart meter devices are installed in a consumer's household and supply energy usage information to a server controlled by the energy provider (which we use as a case study in this chapter for illustrative purposes). This

could be for their own benefit, such as by stealing electricity from the grid. On the other hand, a malicious service provider may wish to clone/mimic a given device's behaviour to impersonate a legitimate user. Such a type of insider attack on the verifier side can be dangerous if the user moves to a different service provider and uses the same device*, which has already been modelled by his/her previous service provider. As a result, it is possible that any one energy provider that was previously enrolled may be compromised, subjecting end-users previously enrolled with the provider to attacks. In this situation – with PUF-based authentication in mind – it is desirable to not require an explicit look-up database of CRPs to perform authentication. Firstly, this type of scheme would maintain forward secrecy, such that if a given verifier is compromised, old messages intercepted by adversaries cannot be decrypted through the derivation of keys as a result of PUF modelling. Second, this scheme would prevent insider attacks (and thus backward secrecy), where a member with access rights at the verifier attempts to build a PUF model using the CRP database in order to compromise a PUF device after it has switched to a new verifier. Both of these situations necessitate the definition of what we refer to in this work as a '*semi-honest verifier*'. With this in mind, we propose a PUF-based authentication scheme where a semi-honest verifier is considered, such that the authenticating server performs a rolling round-based authentication on the device, and, if removed as the authenticating body, is unable to determine future keys required for authentication, thus establishing backward secrecy.

5.1.2 Contributions

The key contributions of this Chapter are as follows:

- We present a novel PUF-based authentication protocol to ensure forward secrecy in the presence of a new potential threat known as a 'semi-honest verifier'. To the best of our knowledge, we are the *first* to consider the concept of a semi-honest verifier for PUF-based authentication schemes and the *first* to apply PUF hardware obfuscation comprehensively at the protocol level.
- We include considerations at the hardware level by utilising both a Strong PUF and DRAM-PUF to provide strong security against ML-MA while maintaining low hardware overhead for the end device. Our scheme is synthesised on a Xilinx Artix-7 FPGA to estimate the required resources and power consumption of an example implementation.
- We perform ML-MA on PUF data simulated with our scheme while considering the presence of the semi-honest verifier, demonstrating the proposed scheme enables security against PUF modelling attacks.
- We provide a formal verification of our protocol, proving its security against a set of realistic adversaries.
- For reproducibility of our results, we provide all code, datasets and proofs from this work in supplementary material as an open resource for the research community (linked in the relevant sections).

*While one given provider (verifier) may supply the smart meter, a consumer may opt to change to a different provider whilst maintaining the same smart meter.

5.1.3 Chapter Organisation

The rest of the chapter is organised as follows. First, Section 5.2 then introduces the proposed scheme. Section 5.3 provides a security analysis of our scheme. Section 5.4 provides a comparison against similar schemes and a discussion of the estimated hardware cost, then finally, the chapter is concluded in Section 5.5. Table 5.1 provides list of notation used throughout the remainder of the chapter.

Table 5.1: *Symbols and Cryptographic Functions*

Symbol	Description
Notation:	
ML-MA	Machine Learning Modelling Attack
PUF	Physically Unclonable Function
CRP	Challenge/Response Pair
Data:	
i	Current (i^{th}) authentication session
$Count$	Counter value
TID	Temporary device identifier
C_S	Strong PUF Challenge
C_M	Memory PUF Challenge
R_S	Strong PUF Response
r_M	Memory PUF Response
R_{Final}	OWF output
K	Key derived from KDF
Δ	Final PUF output for authentication
Cryptographic Functions:	
KDF	Key Derivation Function
SKE.Enc	Symmetric Encryption
SKE.Dec	Symmetric Decryption
OWF	One-Way Function
PRNG	Pseudo Random Number Generator
$\mathsf{PUF_S}$	Strong PUF
$\mathsf{PUF_M}$	Memory PUF

5.2 Proposed Scheme

This section describes our proposed PUF-based privacy-preserving mutual authentication protocol for ensuring security against ML-MA. Before describing our protocol, we describe the on-device PUF requirements, a brief overview of the adversary model and some underlying assumptions for the proposed scheme.

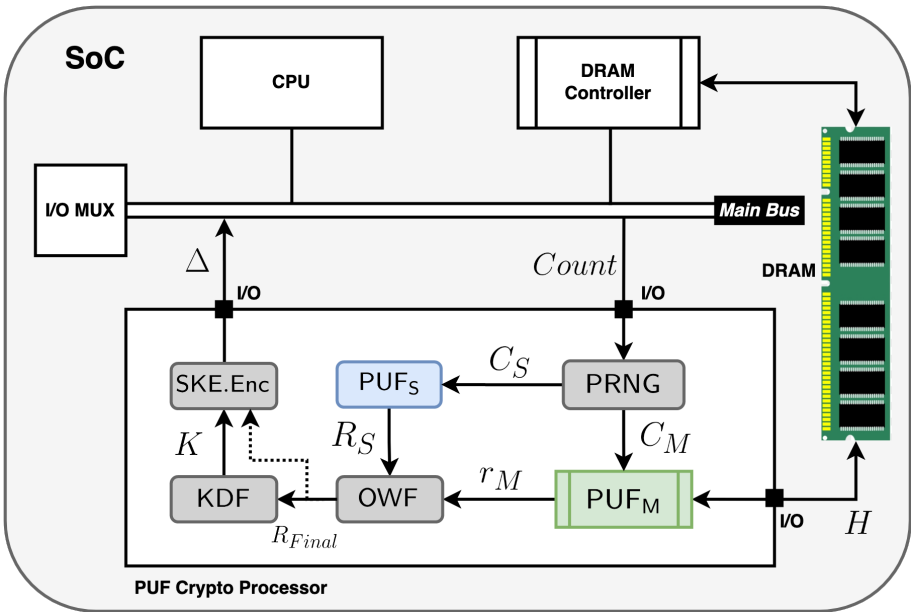

Figure 5.1: *Example of obfuscated PUF processor architecture for the proposed scheme*

5.2.1 PUF Requirements on Device

To support the PUF function of the device for our proposed scheme, we specify the requirement for a strong PUF (Arbiter-PUF), DRAM-PUF and OWF (described in Sections 2.8.1 and 4.2.1) to derive a large number of unique CRPs for use in device authentication. The entire combination of these units contributes to a whole 'PUF'-like function, as it takes an original input challenge (including a DRAM-PUF challenge for OWF configuration) and outputs a final response message. Figure 5.1 demonstrates the interaction of the PUF for our proposed scheme. Firstly, a strong PUF (in our case, an Arbiter PUF) PUF$_S$ receives a randomly generated and unique k-bit input challenge to output a 1-bit response, which is repeated j times (j denoting the required final response length) to generate a response message of j length, R_S, and is stored in a temporary buffer. R_S is used as an input message to the OWF, which outputs deterministically dependent on a configuration value, r_M. r_M is generated by the DRAM-PUF, PUF$_M$, which is synthesised from a raw DRAM response, H[†]. The final response R_{Final} is generated as the output of the OWF, providing a strong cryptographic value for deriving a unique and unpredictable rolling key to be used at the protocol level. Using a key derivation function, KDF, the key K can be generated from R_{Final} to produce the final cryptographic value Δ using symmetric encryption algorithm SKE.Enc for authentication purposes. In this work, we utilise a PRF as a KDF given the assumption of a highly-uniform OWF output, which is portrayed in Figure 5.1 and discussed practically in Section 5.4.

[†]PUF$_M$ here is a (DRAM) memory controller configured for PUF use, where C_M is translated into an input signal to the actual memory and reads back the raw response, H.

5.2.2 Adversary Model

We specify a threat model which includes a typical protocol-level adversary, PUF-level adversary and a new adversary with the new semi-honest verifier in mind. For this, we separate each adversary into one of three types and define our threat model as follows:

- The Type 1 adversary consists of the typical Dolev-Yao model, which is capable of eavesdropping on the network between the smart meter and the energy service provider. This type of adversary can change messages and block messages from device to server or server to device [19].
- The Type 2 adversary consists of an ML-MA capable PUF adversary. This adversary is stronger than the type 1 adversary as they can perform the same actions while also being capable of collecting arbitrary numbers of PUF CRPs and performing ML-based modelling attacks on the PUF in an attempt to clone a PUF. Given enough CRP data for a single APUF, they can successfully model the PUF regarding the work in [74].
- Finally, we consider a Type 3 adversary, which we denote as our 'semi-honest adversary'. This attacker is assumed to operate with elevated privileges at the server-side (service provider) and has complete access to device-to-server messages and data stored within the server database. This adversary is also ML-MA capable and may attempt to model the PUF using CRP information gained. This adversary may also operate as a PUF manufacturer, who has access to the individual PUF during manufacturing in order to model the PUF on its own.

5.2.3 Assumptions

We make the following assumptions regarding the operating environment of our proposed scheme:

- Any adversary may have physical access to the PUF device and, therefore, may access the input and output interface of the overall PUF. This is to say, they may apply arbitrary challenges (combinations of $\mathsf{PUF_S}$ and $\mathsf{PUF_M}$ challenges) and read the final output responses of the overall PUF scheme to collect a dataset of CRPs.
- Any adversary knows the architecture of the device hardware (including the obfuscation scheme).
- A semi-honest adversary at the manufacturing level may collect individual PUF CRPs; for example, a manufacturer of the strong PUF may collect a set of CRPs from the individual $\mathsf{PUF_S}$. Likewise, the device manufacturer may collect a set of $\mathsf{PUF_M}$ CRPs.
- A semi-honest adversary at the server level (during enrollment/authentication) may collect all data sent to and from the device during use with them as a given provider. If the service provider changes, they can no longer access future data sent between devices and the new service provider.

5.2.4 Proposed Authentication Protocol

As with most PUF-based authentication protocols, our proposed scheme consists of two key processes: an Enrollment Phase, where a given device is synchronised with the verifier for the first time, and an Authentication Phase, where a deployed device initiates an authentication attempt with a verifier.

5.2.4.1 Enrollment Phase

Figure 5.2 shows the enrollment procedure for the proposed scheme. All actions taken during enrollment are deemed to be performed honestly in a secure location. A verifier requires a unique identifier (UID) in order to track up-to-date authentication information for multiple different devices; therefore, a device first sends UID to the verifier. The verifier then generates and sends two counters $Count^x$ and $Count^y$ to the Device for the i^{th} (in this case, the first) authentication round, which will be utilised to maintain synchronisation between the Device and Verifier across multiple sessions. For the number of final response bits required, n, as Strong PUF challenges are binary numbers, the Device first generates a challenge C_S for the Strong PUF $\mathsf{PUF_S}$ using its Pseudo Random Number Generator (PRNG) with $Count^x$ as the seed value. c_S is then input to $\mathsf{PUF_S}$ to generate response bit/s r_S. r_S is then appended to the main response R_S^i (or forms the first value if it is the first iteration). After each j iteration up to n, $Count^x$ is incremented to provide a fresh c_S. Next, a DRAM-PUF ($\mathsf{PUF_M}$) challenge C_M^i is selected pseudo-randomly based on $Count^y$.
Due to the non-numerical nature and relatively finite set of possible memory PUF challenges (e.g., 0xFF000000, Zero pattern), we define a Select algorithm to generate suitable challenges without the need to store explicit possible challenges (Algorithm 5). Unique DRAM-PUF challenges consider two variables, the first being *base address*, denoting the area of memory to generate the response from, and *challenge pattern*, denoting the bit pattern to write to the chosen location, where there are three possible patterns: all zeroes, all ones and a mixed checkerboard pattern of zeroes and ones. Therefore – similar to the challenge selection process for the strong PUF – $Count^y$ is used to seed PRNG to provide the random value q. The modulus of q for the A number of available base addresses, a, is generated to provide a base address value between 0 and a. The same is computed for the modulus of q for the P (three), a number of possible challenge patterns. Both integers a and p are then concatenated and returned, enabling the corresponding challenge to be applied to the DRAM-PUF by the memory controller to generate r_M^i. Now, the OWF is configured with r_M^i, and the final output R_{Final}^i is generated from the input message R_S^i. Finally, the Device stores both $Count^x$ and $Count^y$ to establish synchronisation with the verifier before sending R_{Final}^i and r_M^i to the Verifier. The Verifier then finally stores $Count^x$, $Count^y$ for synchronisation, UID for device identification, R_{Final}^i and r_M^i to enable authentication during the first round of the authentication phase.

5.2.4.2 Authentication Phase

Figure 5.3 shows the authentication procedure of the proposed scheme, which has been broken down into a series of sequential steps.

- **Step 1:** First, a device initiates an authentication request with a given verifier by

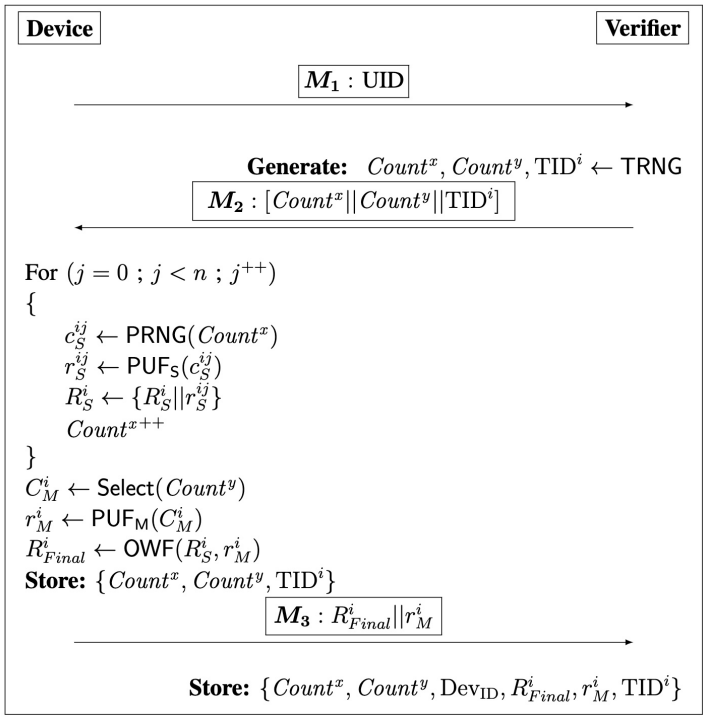

Figure 5.2: *Enrollment phase*

sending a unique Temporary Identification Number, TID^i, where i denotes the current authentication round.

- **Step 2:** After checking that TID^i matches one stored, the verifier then generates the unique key K^i with a Key Derivation Function, KDF, using the final response established in the previous authentication round, R_{Final}^i, as input. Then, the nonce N_V is generated using a True Random Number Generator (TRNG) to enable the device to detect the presence of a replay attack later. Next, using the DRAM-PUF response stored from the enrollment phase/previous round r_m^i to configure OWF, the verifier generates a verification value Res_V consisting of a concatenation of K^i, $Count^x$, $Count^y$ and N_V, which is then sent alongside $Count^x$, $Count^y$ and N_V to the device.

- **Step 3:** Upon receiving the message M_2, the device first assigns a temporary value $Temp^x$ to $Count^x$ for use verifying Res_V later as $Count^x$ will be updated prior to verification. Then, for each j bit required in the final response bit length up to n bits, the strong PUF CRPs are generated. Specifically, the strong PUF challenge c_S^{ij} is generated using PRNG using $Count^x$ as the seed, followed by the response value r_S^{ij} being output from $\mathsf{PUF_S}$ with c_S^{ij} as input. The response bit r_S^{ij} is then appended to the larger response R_S^i, where finally $Count^x$ is incremented before repeating for the j^{th} iteration. After n iterations, a full response message R_S^i of size n will have been generated. Now, following Algorithm 5, the DRAM-PUF challenge C_m^i is generated and applied to the DRAM-PUF via the PUF controller to synthesise the response $\tilde{r_M^i}$. $\tilde{r_M^i}$ is then used to configure the OWF, where R_S^i is used as the input message to output

Algorithm 5: $\mathsf{PUF_M}$ Challenge Selection Algorithm

Select

Input: $Count^x$: Integer
Output: C_M: $\mathsf{PUF_M}$ Challenge
Data: A: Num Base Addresses
$P = 3$: Challenge Patterns (Zero, One, Checkered)

$q \leftarrow PRNG(Count^2)$
$a \leftarrow q \bmod A$
$p \leftarrow q \bmod P$
$C_M \leftarrow \{a||p\}$
return C_M

$\tilde{R^i_{Final}}$. Next, the device generates the key K^i from KDF configured with $\tilde{R^i_{Final}}$. Now, the device generates its own verification value $\tilde{Res}_v$ through concatenating K^i, $Temp_x$, $Count^y$ and N_V using $\tilde{r^i_M}$ for configuration. Providing the verifier is legitimate and was knowledgeable of the correct R_{Final} value for the current session and thus able to generate the correct key K^i, and the counter values $Count^x$ and $Count^y$ and nonce N_V are correct, the device can authenticate the verifier. The device will abort the authentication attempt if any one value is incorrect. $Count^y$ is then incremented for generating the next DRAM-PUF response. The process of strong PUF and DRAM-PUF response generation is repeated to produce a new R_{Final} and r_M pair for use in the next (the $i + 1^{th}$) authentication round. The device then symmetrically encrypts the secrets R^{i+1}_{Final} and r^{i+1}_M to produce Δ. Now, the device computes the verification value Res_D using its OWF, with the concatenation of Δ, $Count^x$ and $Count^y$ as the input, configured with the DRAM-PUF response for the $i + 1^{th}$ round, r^{i+1}_M. A new temporary identifier TID^{i+1} is generated using OWF configured with r^{i+1}_M using the concatenation of K^i and TID^i. Finally, the device stores TID^{i+1}, $Count^x$ and $Count^y$ before sending Δ and Res_D in message M3 to the verifier.

- **Step 4:** Now, the verifier decrypts Δ using the shared key K^i to obtain R^{i+1}_{Final} and r^{i+1}. The device computes its own verification value $\tilde{Res}_D$ with the OWF taking Δ, $Count^x$ and $Count^y$ as input, configured with the now known r^{i+1}_M. If $\tilde{Res}_D$ and Res_D do not match, authentication is aborted by the verifier. Upon successfully matching, the verifier increments $Count^x$ and $Count^y$ the generates the new temporary identifier TID^{i+1} with OWF, configured with r^{i+1}_M with Ki^i concatenated with the previous temporary identifier TID^i as the input message. Finally, the verifier updates R^i_{Final} and r^i_M values with R^{i+1}_{Final} and r^{i+1} respectively before storing TID^{i+1}, $Count^x$, $Count^y$, R^{i+1}_{Final} and r^{i+1} in the database ready for the next authentication round with the device.

5.3 Security Evaluation

In this section, we perform system-level evaluations to assess the proposed scheme's security properties when considering the assumptions and adversary model presented in Section

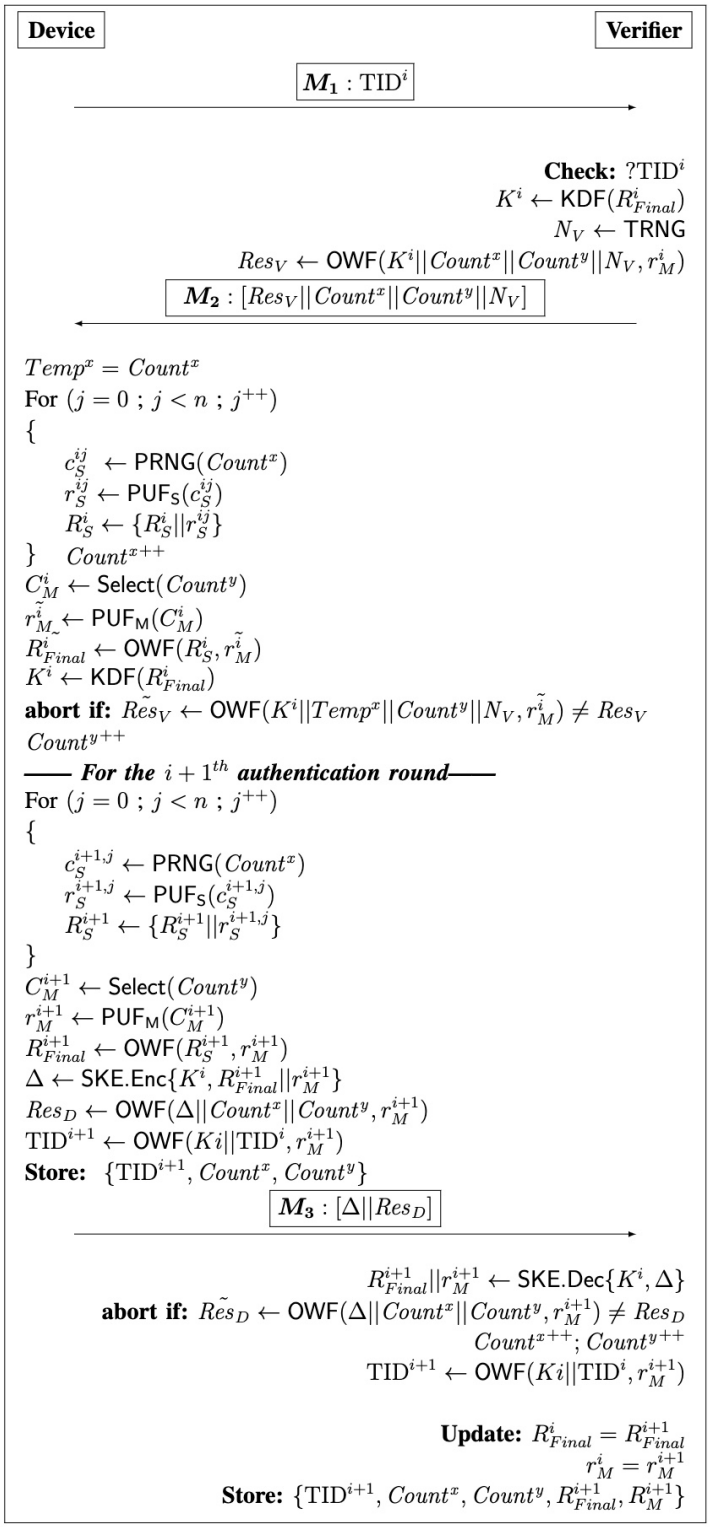

Figure 5.3: *Proposed authentication protocol*

5.2.2. We provide the full proof at the following link: `https://drive.google.com/drive/folders/1ICWO-wYQy72x0VPYrtQwYpz02ZLug4Sj?usp=drive_link`. Additionally the all code can be accessed at [66]: `https://doi.org/10.15131/shef.data.27095617.v1`. Finally, the dataset generated and used can be accessed at [62]: `https://doi.org/10.15131/shef.data.26977237.v1` .

5.3.1 System-Level Security Evaluation

Modelling Methodology

Due to the assumption that an adversary knows the architecture of the device hardware and thus can perform their own computations of the scheme and to align the validation more closely with the established ML-MA PUF literature, we determine the input features of the attack to be a combination of both strong PUF challenge and unique memory PUF challenge, with the one-bit strong PUF output being the binary prediction value (C_S, C_M and r_S respectively as described in Table 5.1). We, therefore, perform the binary classification of strong PUF output using both the logistic regression (LR) and multi-layer perceptron (MLP) attacks as described in both [74] and [1], respectively. We use raw accuracy as an evaluation metric because the dataset is balanced (low bias to either 0 or 1 output) and because, unlike common binary classifiers where output may be considered 'positive' or 'negative', in the case of single bit PUFs, either observed output is desirable. Therefore, precision and recall are not required as evaluation metrics for determining the performance of the attacks.

PUF Dataset

We generated a set of CRPs by simulating a strong PUF and a lightweight OWF configured using real memory PUF data, which is adapted from the work proposed by Augot et al. [3]. We tested a 16-bit and a 32-bit Arbiter PUF using the Pypuf Python library [89] to simulate the strong PUF of our scheme, generating the challenges C_S and responses r_S. The OWF was also developed in Python, and the memory PUF responses used to configure the OWF were obtained from a real DRAM Latency PUF test bed as described by Miskelly et al. [68]. We assume that an attacker is unable to predict or gain knowledge regarding the memory PUF; however, due to the relatively low number of novel memory PUF configurations (and thus responses), we allow our attacker to determine the category of the memory PUF challenge in the case that $Count^y$ is intercepted and determined using the PRNG. The attacker, therefore, gains the categorical data of the memory PUF challenge, i.e., memory location a, pattern 0. In our DRAM-PUF dataset, we utilised twelve unique DRAM-PUF responses for OWF configuration by using four unique locations: *a, b, c, d* and three unique input patterns: *all zeroes, all ones* and *checkerboard*. Due to the categorical nature of this feature, we performed one-hot encoding to transform each DRAM-PUF category into a set of twelve-digit binary features to denote each category separately to normalise the data for training. Therefore, the dataset used for training consisted of a set of strong PUF challenges combined with a one-hot encoded value denoting the DRAM-PUF response used to configure the OWF, providing unique challenge samples, even if an exact individual strong PUF challenge is repeated. Each challenge sample was paired with a corresponding binary value of 1 or -1 [‡] response as

[‡]**-1 is used instead of 0 for normalisation of data for model training.**

measured from our simulation. Both the datasets and code is provided in Appendix D in the supplementary material.

Effect of scheme on ML-MA

Figure 5.4 outlines the performance of both the LR and MLP attacks on both the 16 and 32-bit Arbiter PUFs when utilised with the proposed scheme. We performed each attack using 5000, 20,000, 50,000, 75,000, 100,000, 250,000 and 500,000 CRPs – denoted N_{CRP} – for training for each PUF type. It was found that the prediction accuracy did not scale with N_{CRP} as is often seen in related works when directly attacking a PUF [74]. While the highest observed prediction accuracy was 59.1% for the LR attack on the 16-stage APUF with $N_{CRP} = 75,000$, the prediction accuracy for the same attack with $N_{CRP} = 500,000$ was only 52.5%. The results indicate that there is no correlation between N_{CRP} and prediction rate, which is likely caused by a degradation in the generalisation ability of the model when combined with the strong non-linearity provided by the OWF when split across the random sub-samples of the entire dataset. The most ideal prediction rate is 50%, which is comparable to a random coin flip. A PUF is considered vulnerable to ML-MA if a prediction rate of 70% or above is achievable, demonstrating the proposed scheme is sufficient to prevent ML-MA.

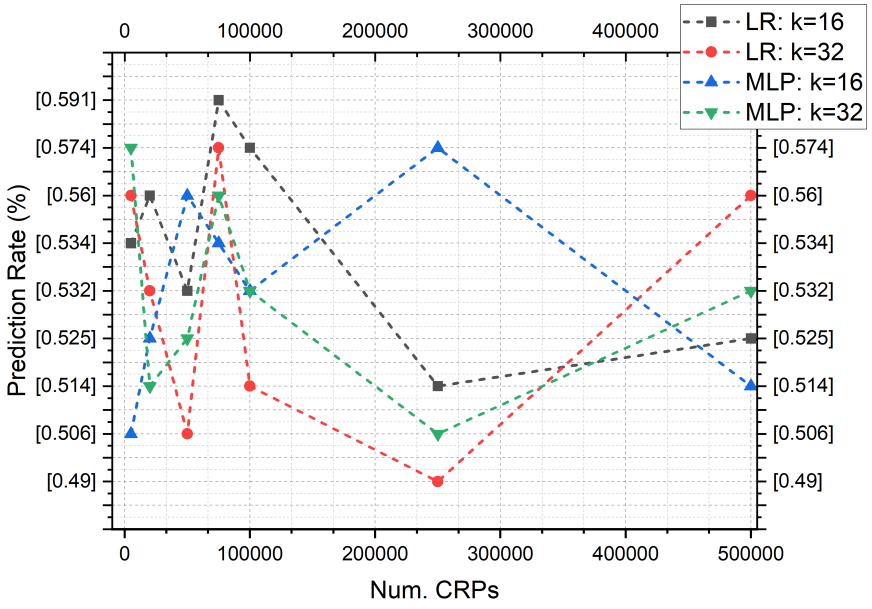

Figure 5.4: *Prediction accuracies for each LR and MLP modelling experiment against the k=16 stage and k=32 stage APUFs when obfuscated with the proposed scheme*

5.3.2 Protocol-Level Security Evaluation

In this section, we discuss the security of our proposed protocol against the Type 1 and Type 2 adversaries defined in Section 5.2.2. For brevity, we only present the details of the proof of

MA-security and security against the Re-Use Model. The full version of each proof is available at `https://drive.google.com/drive/folders/1ICWO-wYQy72x0VPYrtQwYpz02ZLug4Sj?usp=drive_link` under 'Chapter 5/Supplementary_Material'.

5.3.2.1 Mutual Authentication Security

To resist the man-in-the-middle attack, we must guarantee that both parties accept the authentication only if the communication message is honestly transferred. The security model captures the security of the authentication protocol as a pair of games $\mathbf{Exp}^{MA}_{\Pi,\mathcal{A}}(\lambda)$ played between a probabilistic polynomial-time adversary $\mathcal{A}$ and a challenger $\mathcal{C}$, where Π denotes the protocol and λ denotes security parameters. In the experiment, the adversary wins the game if it causes a **clean** session to accept Res_D, then $\mathbf{Exp}^{MA}_{\Pi,\mathcal{A}}(\lambda)$ outputs 1.

Adversary Queries: Here, a Type 1 adversary $\mathcal{A}$ can control and eavesdrop the communication channel between the device D and verifier V, and compromise the shared secrets with the following queries:

1. $Create(i, s)$: allows $\mathcal{A}$ to initialize new session π^s_i, where i is the session index and s is session owner.
2. $Send(m, i, s)$: allows $\mathcal{A}$ to send arbitrarily message m to $\mathcal{C}$ in session π^s_i.
3. $Corrupt(i) \rightarrow K^i$: allows $\mathcal{A}$ to reveal session key K^i.
4. $StateReveal(i, s, c) \rightarrow \pi^s_i$: allows $\mathcal{A}$ to reveal the internal state of π^s_i when challenge is $c^{'}$.

Definition 1 *(Cleanness predicate): A session π^s_i in the modeling experiment described above is **clean** if $Corrupt(i)$ was not issued, and $StateReveal(i, s, c)$ was not issued, and for all $j, s^{'}$ such that $c^{'} = c$, $StateReveal(j, s^{'}, c^{'})$ was not issued.*

Definition 2 (*Mutual Authentication Security*) *An authentication protocol Π holds the mutual authentication security against man-in-the-middle attack if for any PPT type 1 adversary $\mathcal{A}$, $\mathbf{Exp}^{MA}_{\Pi,\mathcal{A}}(\lambda)$ is negligible.*

Theorem 1 *Let PUF be a $(N_{CRP}, \epsilon_{puf})$-secure PUF, OWF be a secure one-way function with $\epsilon^{revs}_{\mathsf{OWF}}$ and $\epsilon^{\mathsf{OWF}}_{\mathsf{OWF}}$. Then our protocol Π holds mutual authentication security that for any PPT $\mathcal{A}$, $\mathbf{Adv}^{MA}_{\Pi}(\mathcal{A})$ is negligible.*

Proof: The goal of $\mathcal{A}$ is to win the mutual authentication game, in which $\mathcal{C}$ accepts the session without matching session. We can divide the proof into two cases: for *device impersonation* and *verifier impersonation*. In case 1, $\mathcal{A}'s$ advantage is $\mathbf{Adv}^{\text{MA,clean, C1}}_{\Pi,\mathcal{A}}$. In case 2, the advantage is $\mathbf{Adv}^{\text{MA,clean, C2}}_{\Pi,\mathcal{A}}$. It is clear that $\mathbf{Adv}^{\text{MA,clean}}_{\Pi,\mathcal{A}} \leq \mathbf{Adv}^{\text{MA,clean, C1}}_{\Pi,\mathcal{A}} + \mathbf{Adv}^{\text{MA,clean,C2}}_{\Pi,\mathcal{A}}$.

Case 1: *Verifier Impersonation*

Game $A_{1.0}$: The original mutual authentication game explained above, we know $\mathbf{Adv}^{\text{MA,clean, C1}}_{\Pi,\mathcal{A}} \leq \mathbf{Adv}_{G_{\mathcal{A}_{1.0}}}$.

Game $A_{1.1}$: Here we introduce an abort event, where $\mathcal{C}$ aborts when $\mathcal{A}$ successfully generates a valid verification value $Res_V^{'}$ which equals Res_V. The advantage of breaking the collision avoidance of OWF is $\mathbf{Adv}^{revs}_{\mathsf{OWF}}$. Thus the probability that $\mathcal{A}$ wins is bounded by the collision security of OWF, therefore $\mathbf{Adv}_{G_{\mathcal{A}_{1.0}}} \leq \mathbf{Adv}^{revs}_{\mathsf{OWF}} + \mathbf{Adv}_{G_{\mathcal{A}_{1.1}}}$.

Game $A_{1.2}$: In this game, we replace K^i by interacting with a KDF challenger. Any $\mathcal{A}$ that can distinguish *Game* 1.1 and *Game* 1.2 can be used to break KDF-security. Thus $\mathbf{Adv}_{G_{A_{1.1}}} \leq \mathbf{Adv}_{G_{A_{1.2}}} + \mathbf{Adv}^{Ind}_{\mathsf{KDF}}$.

Game $A_{1.3}$: In this game, $\mathcal{A}$ obtains the right response R^{i+1}_{Final} and r^{i+1}_M with advantage of $\mathbf{Adv}^{unpredict}_{PUF}$. Thus $\mathcal{A}$ can add advantage bounded PUF: $\mathbf{Adv}_{G_{A_{1.2}}} \leq \mathbf{Adv}_{G_{A_{1.3}}} + \mathbf{Adv}^{unpredict}_{PUF}$.

Game $A_{1.4}$: In the game, π^s_i will only accepts Rev_D from an honest verifier. From **Game** $A_{1.2}$, we know that $\mathcal{A}$ cannot produce valid K^i, neither Res_V from **Game** 1.1 with non-negligible advantage. Thus we know $\mathbf{Adv}_{G_{A_{1.4}}} = 0$.

Case 2: *Device Impersonation*

Game $A_{2.0}$: The original mutual authentication game explained above, we know that $\mathbf{Adv}^{\text{MA,clean, C2}}_{\Pi,\mathcal{A}} \leq \mathbf{Adv}_{G_{\mathcal{A}_{1.0}}}$.

Game $A_{2.1}$: Here we introduce an abort event, where $\mathcal{C}$ aborts when $\mathcal{A}$ successfully generates a valid verification value $\Delta||Res_D$ corresponding to the session. The advantage of breaking the collision avoidance of OWF and SKE is $\mathbf{Adv}^{revs}_{\mathsf{OWF}}$ and $\mathbf{Adv}_{\mathsf{SKE}}$. Thus the probability that $\mathcal{A}$ wins is bounded by the security of OWF and SKE, $\mathbf{Adv}_{G_{\mathcal{A}_{2.0}}} \leq \mathbf{Adv}^{revs}_{\mathsf{OWF}} + \mathbf{Adv}_{\mathsf{SKE}} + \mathbf{Adv}_{G_{\mathcal{A}_{2.1}}}$.

Game $A_{2.2}$: In this game, $\mathcal{A}$ obtains the K^i similar to **Game** $A_{1.2}$ and the right response similar to **Game** $A_{1.3}$, adding advantage bounded by the security of KDF and unpredictability of PUF: $\mathbf{Adv}_{G_{\mathcal{A}_{2.1}}} \leq \mathbf{Adv}^{Ind}_{\mathsf{KDF}} + \mathbf{Adv}^{unpredict}_{PUF} + \mathbf{Adv}_{G_{\mathcal{A}_{2.2}}}$.

Game $A_{2.3}$ In the game, π^s_i will only accepts $\Delta||Rev_D$ from an honest device. From **Game** 2.2, we know that $\mathcal{A}$ cannot produce valid Δ, neither Res_V from **Game** $A_{1.1}$ with non-negligible advantage. Thus we know: $\mathbf{Adv}_{G_{A_{2.3}}} = 0$.

5.3.2.2 Security Against Type 3 Attacker

In [75], Rührmair et al. give an extension setting of the "PUF re-use model", which allows adversaries multiple access to PUFs, for example, before, after or between protocol executions. As we know, it is the strongest model because the adversary is allowed physical or evaluation access to the PUF instance. In our attack scenario, as defined in Type 2 adversary, $\mathcal{A}$ has the CRP access to the PUF, and the number of CRPs the adversaries can read out is

arbitrary. Here we present the security analysis against the Type 3 attacker. In many scenarios, the server or verifier may also be untrustworthy. In this case, we also must guarantee the unavailability of the whole PUF model for adversaries. In other words, in our protocol, the server may only invoke the authentication functionality of the device rather than access the real CRPs of certain PUFs. As defined in the Type 3 adversary at the server level, $\mathcal{A}$ has every permission we give to the verifier, including obtaining the registration information from the enrollment phase. At the manufacturing level, $\mathcal{A}$ has collected sufficient CRPs of PUF_S (not PUF_M). The goal of $\mathcal{A}$ is to give the session key $K^{(i+1)}$ for next round $i+1$.

5.3.2.3 Security Against Semi-honest Adversary

In many scenarios, the server or verifier can also be untrustworthy. In this case, we also need to guarantee the unavailability of the whole PUF model for adversaries. In other words, in our protocol, the server can only invoke the authentication functionality of the device rather than access the real CRPs of certain PUF. As defined in Type 3 adversary at the server level, $\mathcal{A}$ has every permission we give the verifier, including obtaining the registration information from the enrollment phase. At the manufacturing level, $\mathcal{A}$ has collected sufficient CRPs of PUF_S or PUF_M. The goal of $\mathcal{A}$ is to give the session key $K_(i+1)$ for the next round $i+1$.

Similar to ***Definition***4, the definition of modeling security provides:

Definition 3 *(Modeling Security Against Semi-honest Adversary) Let* Π *be an authentication protocol. For a clean predicate, and a PPT algorithm* $\mathcal{A}$*, we define the advantage of* $\mathcal{A}$ *in the Semi-Honest Modeling game to be:* $\mathbf{Adv}^{SH}_{\Pi,\mathcal{A},clean}(\lambda) = |Pr[\mathbf{Exp}^{SH}_{\Pi,\mathcal{A}}(\lambda) - \frac{1}{2}]|$*. We say that* Π *is Modeling-secure against re-use model if for all PPT* $\mathcal{A}$*,* $\mathbf{Adv}^{RU}_{\Pi,\mathcal{A}}(\lambda)$ *is negligible in parameter* λ*.*

Proof We separate the security proof in two phases. In the first phase, $\mathcal{A}$ tries to obtain the real final response, and it can reverse the OWF and obtain R^i and r^i_M with the advantage of $Adv^{revs}_{OWF}(\mathcal{A})$. In the second phase, with N_{CRP} CRPs, $\mathcal{A}$ can model the PUF with $Adv^{N_{CRP}}_{PUF}$. Therefore $\mathcal{A}$ has two goals: reverse the OWF and model the PUF, described as follows.

Definition 4 *(Modeling Security) Let* Π *be an authentication protocol. For a clean predicate, and a PPT algorithm* $\mathcal{A}$*, we define the advantage of* $\mathcal{A}$ *in the Re-Use Modeling game to be:* $\mathbf{Adv}^{RU}_{\Pi,\mathcal{A},clean}(\lambda) = |Pr[\mathbf{Exp}^{RU}_{\Pi,\mathcal{A}}(\lambda) - \frac{1}{2}]|$*. We say that* Π *is Modeling-secure against re-use model if for all PPT* $\mathcal{A}$*,* $\mathbf{Adv}^{RU}_{\Pi,\mathcal{A}}(\lambda)$ *is negligible in parameter* λ*.*

Proof We separate the security proof in two phases. In the first phase, $\mathcal{A}$ tries to obtain the real final response and it can reverse the OWF and obtain R^i and r^i_M with the advantage of $Adv^{revs}_{OWF}(\mathcal{A})$. In the second phase, with N_{CRP} CRPs, $\mathcal{A}$ can model the PUF with $Adv^{N_{CRP}}_{PUF}$. Therefore $\mathcal{A}$ has two goals: reverse the OWF and model the PUF, described as follows.

Phase 1: $\mathcal{A}$ generates N_{CRP} pairs of random number $Count^{x'}_{idx}$, $Count^{y'}_{idx}$ and queries $Send(Count^x, Count^y, i, s)$, then $\mathcal{A}$ will the final response R^i_{Final}. Then $\mathcal{A}$ invokes $PRNG(\cdot)$ and $Select(\cdot)$ to generate corresponding challenges c_s and c_M following the exactly same steps in the authentication phase. In the end, $\mathcal{A}$ tries to reverse the OWF to obtain the

Table 5.2: *Comparison of PUF Protocols*

	P_1	P_2	P_3	P_4	P_5	P_6	P_7
Yu et al. [93]	X	✓	✓	X	✓	✓	✓
Gu et al. [33]	X	✓	✓	X	✓	✓	X
Gope et al. [31]	✓	✓	✓	✓	✓	✓	X
Ebrahimabadi et al. [21]	X	X	✓	X	X	X	✓
Chatterjee et al. [13]	X	✓	X	X	✓	X	✓
Modarres et al. [2]	✓	✓	✓	✓	✓	X	✓
Proposed Scheme	✓	✓	✓	✓	✓	✓	✓

✓: Yes; X: No; P_1: **Privacy** P_2: **Mutual Authentication** P_3: **ML-MA Resilience** P_4: **Scalability** P_5: **No CRP Database Required** P_6: **No Third Party Required** P_7: **No NVM Required (On device)**

true responses of PUF_s and PUF_M, namely r_s and r_M. As discussed above, the advantage of reversing OWF for one time can be obtained: $\mathbf{Adv}_{OWF}^{revs} = |\epsilon_{OWF}^{revs} - \frac{1}{2}|$. Advantage of reversing OWF for N_{CRP} time is $\mathbf{Adv}_{OWF}^{revs_{N_{CRP}}} = |{\epsilon_{OWF}^{revs}}^{N_{CRP}} - \frac{1}{2}^{N_{CRP}}|$.

Phase 2: After phase 1, $\mathcal{A}$ will have N_{CRP} CRPs, which can be used as training data to model PUF_S and PUF_M. The advantage of this game is determined by the number of CRPs, which can be interpreted as the accuracy of the machine learning model, defined as $\mathbf{Adv}_{PUF_S}^{modeling} = |\mathbf{Acc}_{PUF_S}^{N_{CRP}} - \frac{1}{2}|$ and $\mathbf{Adv}_{PUF_M}^{modeling} = |\mathbf{Acc}_{PUF_M}^{N_{CRP}} - \frac{1}{2}|$. Apparently, the advantage of $\mathcal{A}$ depends on the number of CRPs and the modeling accuracy. Therefore the probability that $\mathcal{A}$ wins is bounded by: $\mathbf{Adv}_{G_{b2}} \leq Min\{\mathbf{Adv}_{PUF_S}^{modeling}, \mathbf{Adv}_{PUF_M}^{modeling}\}$. We know the success of , the overall advantage can be obtained by $\mathcal{A}$ is:$\mathbf{Adv}_{PUF}^{RU} = {\epsilon_{OWF}^{revs}}^{N_{CRP}} * Min\{\mathbf{Adv}_{PUF_M}^{N_{CRP}}, \mathbf{Adv}_{PUF_S}^{N_{CRP}}\}$.

As we know, the more $CRPs$, the closer $\mathbf{Adv}_{PUF}^{N_{CRP}}$ is to $\frac{1}{2}$. Thus we can set it to $\frac{1}{2}$ to get the supremum. Since the OWF we use in the protocol is secure, thus $\mathbf{Adv}_{OWF}^{revs}$ is negligible; therefore, the N_{CRP} power of it is also negligible.

5.4 Discussion

In this section, we provide a comparison of the overall features of our scheme against other PUF-based authentication protocols and discuss the estimation of the hardware resource requirement to deploy the required functionality on a given device. We choose protocols for comparison based on a few factors. Primarily, we determine protocols which have given a focus on the PUF/system-level implementation and implications (for example, discussing amount of NVM required for key storage). We also compare protocols specifically designed for lightweight authentication in IoT. This selection captures both the state-of-the-art as well as some foundational protocols within this research space.

5.4.1 Comparison Against Other Protocols

A comparison of the proposed scheme against similar PUF-based authentication protocols is shown in Table 5.3. Against desirable properties, each protocol contains one or more deficits which place restrictions on their application. For example, the protocol proposed by Ebrahimabadi et al. [21] satisfies ML-MA; however, it does not support privacy or mutual authentication, in addition to requiring a specific CRP database to be stored on the server. Gope et al. propose a protocol in [31] which satisfies the most desirable properties; however, it requires some NVM to store key data, making it vulnerable to probing. Modarres et al. propose an effective and scablable protocol designed for the Internet of Medical Things (IoMT) systems. While satisfying most key properties, there is an increased complexity in this protocol as it requires more than two parties.

5.4.2 Estimated Cost of Implementation

In order to estimate the total hardware resource requirement for the proposed scheme, we refer to Figure 5.1 to determine the cost of each individual component required for the function of the PUF processor on a Xilinx Artix-7 FPGA. While the FPGA based components were fully synthesised, we refer to this as an *estimate* for a few distinct reasons. Firstly, in Figure 5.1, error correction is assumed to be included in $\mathsf{PUF_S}$. The overhead required for a given error correction method can vary greatly depending on the error rate of the implemented PUF. As Arbiter PUFs become more prone to noisy outputs as they increase in size (and if multiple XOR's are implemented), we make an assumption of a low error rate due to proposing the use of small APUFs (16 or 32-bit). Highly reliable APUF implementations have been proposed in the wider literature, such as in [37], where an error rate of $< 10^{-9}$ is reported, therefore a light error correction technique such as reported in [38] would be suitable. For each other component, while different types of each could be used interchangeably, we synthesised an example of our PUF processor design using Xilinx Vivado Design Suite to determine an estimation of the required resources for each on an FPGA, the total resource requirements in Look-Up Tables (LUTs) and D-Flip-Flops (DFFs) and Block RAMs (BRAMs). These estimates are listed in Table 5.3. The FPGA layout for our design can be seen in Figure 5.5. For the DRAM-PUF, no significant restriction on the size of memory is required as an increased size of available memory only provides more unique values for configuring the OWF. To ensure a sufficient amount of novel memory PUF responses, however, we state a minimum of 16KB of memory should be required, as the OWF used in preparing our dataset required 8KB responses for configuration. Given that three unique responses can be synthesised from one memory location (0's, 1's, mixed), 16KB would enable six unique configurations whilst leaving room for disregarding potentially unreliable sections of the available memory. Furthermore, a DRAM-PUF implementation requires no additional FPGA resources as PUF data and (repeated measurements for error correction) are stored in the memory itself (hence 0 BRAMs required) and only require modification of the memory controller in runtime for PUF function. For our reference implementation on the FPGA, we included the following configuration for each key block as follows:

- **Strong PUF:** We use 32-bit Arbiter PUF as a strong PUF.
- **PRNG Functionality:** We implemented a 32-bit LSFR as PRNG, specified by its

feedback primitive polynomial $x^{32} + x^{22} + x^2 + x^1 + 1$.

- **One-Way Function:** A light OWF with inspiration from the work in [3], with a 64-bit input and output.
- **Symmetric Encryption:** Uses the hardware-efficient SIMON block cipher [5].
- **KDF:** For resource-efficiency reasons, we implement PRF instead of KDF thanks to our a highly-uniform OWF output assumption (also reflected in Section 2.8.2). We use the lightweight Keccak-f[200] hash function as PRF [8] as it can be used for pseudorandom number generation.

Table 5.3: *Estimated Total Cost of Our Proposed Scheme on FPGA*

Components	**LUTs**	**DFFs**	**BRAMs (KB)**	**Power Cons. (W)**
- APUF: 32-bit **- LSFR PRNG:** 32-bit **- OWF:** 64-bit **- PRF:** Keccak-f[200] **- SIMON Cipher:** m = 64-bit, n = 32-bit	785	898	0	0.203

5.5 Summary

In this chapter, we proposed a privacy-preserving PUF-based authentication protocol, including hardware considerations for PUF obfuscation in order to prevent ML-MA on PUFs. We presented the novel concept of a 'semi-honest verifier' for PUF-based protocol schemes, where a verifier is assumed not always to be acting legitimately during the life cycle of an authenticating device. Through experimental validation, both theoretically and practically, we prove that our scheme not only prevents ML-MA on PUFs in the presence of this new threat but that the scheme also provides desirable properties such as privacy, forward secrecy and mutual authentication.

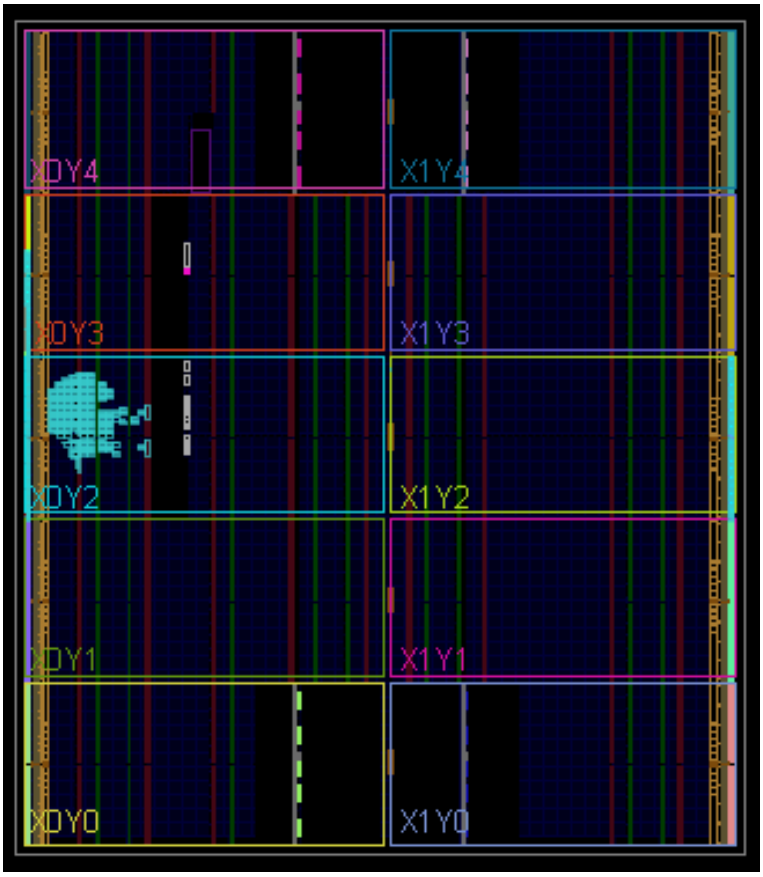

Figure 5.5: *Layout of proposed scheme synthesised on an Artix-7 FPGA*

Chapter 6

Conclusion and Future Work

In this chapter, we summarise the contributions of this thesis, including a summary of the experiments carried out and how they support the hypotheses presented in Chapter 1. Finally, we discuss the open questions that lead to this research's potential future directions.

6.1 Summary of Chapters

DRAM-PUFs are a highly promising technology for enabling strong identity and authentication for low-power/resource-constrained applications. With DRAM being ubiquitous in computer systems, it is imperative to investigate practical applications of DRAM-PUFs for enabling security through hardware Root-of-Trust (RoT); however, there exist limitations to the state-of-the-art approaches. Overall, we identified the following issues:

- Due to the noisy nature of DRAM-PUFs, some current approaches exist which utilise Convolutional Neural Networks (CNNs) for classifying the responses for authentication purposes. These approaches, however, suffer from various limitations as identified in Section 3.1.1, such as model complexity, which reduces the applicability of the techniques to resource-constrained systems. Additionally, current approaches only investigate the classification of individual DRAM-PUFs per CNN model, limiting scalability where multiple PUFs may need to be scrutinised by a given verifier.

- Due to the issue of PUF noise, error correction techniques are most often required, necessitating the storage of publicly accessible helper data. It has been shown that adversaries may exploit helper data to mount modelling attacks on PUFs.

- Current CNN approaches rely on physically mapped DRAM-PUF data such that physical features are encoded in the response image (to capture charge leakage behaviour, etc.). This approach complicates enrollment and potentially leaks additional information to attackers who may wish to model the leakage behaviour of the DRAM regions used to generate responses.

- Strong PUFs are suitable for CRP-based authentication due to their generally exponentially large CRP space; however, they are very vulnerable to Machine Learning Modelling Attacks (ML-MA). Obfuscation methods have been proposed to counter this

vulnerability; however, they have various limitations, including high resource cost and vulnerable Non-Volatile Memory (NVM) requirements. These issues are discussed in Section 4.1.

- Current approaches which aim to utilise DRAM-PUFs for CRP-based authentication are currently restricted in scalability due to the limited unique CRP space available. When discarding a CRP per use, a DRAM-PUF would need to be recalled often, which is highly undesirable for many IoT applications. This is highlighted in Section 4.1.
- Protocol-level integration of DRAM-PUFs is largely overlooked, limiting the realistic application of DRAM-PUFs which exploit their intrinsic benefits as cryptographic primitives. We identify the current limitations in Section 5.1.

To address the key issues identified in this thesis, we presented three experimental studies and evaluated their performance, the hypotheses of which were proposed in Section 1.1. The overall goal of this work was to develop methods for enhancing authentication processes for resource-constrained settings using DRAM-PUFs.

Chapter 3 considered the problems related to current CNN-based authentication using DRAM-PUFs. This work is the *first* to introduce the novel concept of a *PUF Phenotype*, whereby a PUF's response is considered as the digital output, including specific features caused by the environment, without regard for any structural information of the PUF. We generated a novel dataset of temperature and voltage variant Latency DRAM-PUF 'Phenotypes', formatted as a set of images. By modifying a VGG-16 CNN, our experimentation showed that exploiting classical classifiers was very effective at scrutinising the DRAM-PUF Phenotypes, reducing the model's overhead by a factor of 10. The findings in this work demonstrated that by fine-tuning model confidence to determine a threshold, it is possible to authenticate the origin of responses with highly effective false positive/false negative rates. These discoveries support our *"first hypothesis: "Computer vision-based authentication can be employed using DRAM-PUFs for authentication in a practically lightweight fashion. A highly acceptable false positive/false negative rate for authentication can be achieved across multiple devices (multi-class classification) for highly noisy responses".*

Chapter 4 explored how DRAM-PUFs can be utilised to obfuscate Strong PUFs to solve scalability issues in DRAM-PUF CRP space and high resource costs for current state-of-the-art obfuscation schemes. Through our experiments, we demonstrated that it is possible and effective to exploit DRAM-PUFs for securely generating cryptographically significant matrices, which can be used to configure lightweight One-Way Functions (OWF). The approach requires fewer FPGA resources (LUTs and DFFs) than the state-of-the-art schemes and proved to be effective at preventing ML-MA. In addition, the proposed scheme could also support various PUF types, making it generic and, therefore not restricted to a single PUF design. *Hypothesis 2: "DRAM-PUF entropy can be exploited to enhance the obfuscation of Strong PUFs for authentication such that required hardware overhead is minimised while resistance against ML-MA remains high"* was therefore satisfied in this Chapter.

Finally, a protocol-level approach for integrating DRAM-PUF-based obfuscation hardware was demonstrated in Chapter 5. The proposed protocol enabled security against ML-

MA when considering a more capable, novel *semi-honest verifier*. An adapted (from that proposed in Chapter 4) obfuscation approach was presented and experimentally verified on a Xilinx Field Programmable Gate Array (FPGA). The results of ML-MA tests and security analyses demonstrate our *third hypothesis: "A privacy-preserving authentication protocol can be designed which tightly integrates a DRAM-PUF-enhanced Strong-PUF obfuscation design in order to provide further resilience against ML-MA, including resilience against adversaries with significantly elevated privileges to collect PUF data and mount attacks."*

6.2 Potential Future Works

The works presented when addressing the hypotheses introduce the following questions, an investigation of which could effectively improve upon the contributions made in this thesis.

Extension of PUF Phenotype Classification/Authentication: In Chapter 3, our presented approach focuses on PUF Phenotype images, which currently consist of relatively large 8KB memory regions. It would be insightful to investigate the impact of using smaller images during training, such as splitting them into four 2KB images or eight 1KB images. This exploration could yield two notable advantages:

- Enhanced Authentication Space: By reducing image size, the unique CRP space for authentication can be increased, improving the overall scalability of the scheme.
- Lower Computational Cost: Smaller images not only reduce the training and execution computational overhead of the DPAN model but also ease the burden of processing and generating Phenotype images on verifying devices.

However, it is crucial to acknowledge a potential weakness with this approach. Smaller Phenotype images might simplify the expected Phenotype image's complexity, potentially making it easier for attackers to deceive the model. Thus, further investigation and experimentation are required to strike a balance between these benefits and risks.

Furthermore, since PUF Phenotypes are 'structure-agnostic´, it becomes feasible to explore their effectiveness in authenticating individual Phenotypes that incorporate multiple types of PUF output and therefore is an interesting avenue for further investigation.

Finally, whilst in this work the group membership was treated as a static property for experimental purposes, it would be entirely possible to retain the PUF characterisation data for each device in a central and highly secure environment and use this to update group membership actively. This approach would require retraining of the model on the new set of group members and then pushing the new model out to all devices. A full exploration of this kind of approach to PUF at the protocol level should form the basis of future work.

Side-channel Analysis of PUF Obfuscation Schemes In Chapters 4 and 5, two similar hardware schemes are presented. Whilst the hardware and power costs were experimentally verified, sophisticated hardware-level attacks such as side-channel and fault injection attacks

are important to consider when increasing the expected capability of the adversary. Exploration into the robustness of the hardware schemes presented in this thesis against these types of attacks is a logical next step for security analysis.

ML-MA Resilience of DRAM-PUF The current literature assumes that Strong PUFs are vulnerable to ML-MA and Weak PUFs are not primarily due to available CRP space. While DRAM-PUFs are generally considered Weak PUFs, as noted in Section 2.4.7, it is sensible to designate DRAM-PUFs as a type of 'Semi Weak' PUF due to the vastly increased CRP space (per-bit) over 'traditional' Weak PUFs (such as Optical/SRAM-PUFs). Due to the (relatively) large CRP space available and the known charge-leakage effects of contiguous bit-cells, it remains an open question as to the difficulty of modelling DRAM-PUF outputs, given knowledge of limited CRPs. A security evaluation at this level should be investigated to enhance trust in DRAM-PUFs as cryptographic primitives further.

Bibliography

[1] Mohammed Saeed Alkatheiri and Yu Zhuang. Towards Fast and Accurate Machine Learning Attacks of Feed-Forward Arbiter PUFs. In *2017 IEEE Conference on Dependable and Secure Computing*, pages 181–187, 2017.

[2] Amir Masoud Aminian Modarres, Nima S. Anzabi-Nezhad, and Maryam Zare. A new puf-based protocol for mutual authentication and key agreement between three layers of entities in cloud-based iomt networks. *IEEE Access*, 12:21807–21824, 2024.

[3] Daniel Augot, Matthieu Finiasz, and Nicolas Sendrier. Improved Fast Syndrome based Cryptographic Hash Functions. In *in Proceedings of ECRYPT Hash Workshop 2007 (2007). URL: http://www-roc.inria.fr/secret/Matthieu.Finiasz*.

[4] Aydın Aysu, Ege Gülcan, Daisuke Moriyama, Patrick Schaumont, and Moti Yung. End-To-End Design of a PUF-Based Privacy Preserving Authentication Protocol. In Tim Güneysu and Helena Handschuh, editors, *Cryptographic Hardware and Embedded Systems – CHES 2015*, pages 556–576. Springer Berlin Heidelberg, 2015.

[5] Ray Beaulieu, Stefan Treatman-Clark, Douglas Shors, Bryan Weeks, Jason Smith, and Louis Wingers. The SIMON and SPECK Lightweight Block Ciphers. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, 2015.

[6] Georg Becker. On the Pitfalls of using Arbiter PUFs as Building Blocks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34:1–1, 08 2015.

[7] E. Berlekamp, R. McEliece, and H. van Tilborg. On the Inherent Intractability of Certain Coding Problems (corresp.). *IEEE Transactions on Information Theory*, 24(3):384–386, 1978.

[8] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, pages 313–314, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[9] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[10] Kathrin Blagec, Georg Dorffner, Milad Moradi, and Matthias Samwald. A Critical Analysis of Metrics used for Measuring Progress in Artificial Intelligence, 2020.

[11] Simona Buchovecká, Róbert Lórencz, Filip Kodýtek, and Jiří Buček. True Random Number Generator based on Ring Oscillator PUF Circuit. *Microprocessors and Microsystems*, 53:33–41, 2017.

[12] Julius Cepukenas, Chenghua Lin, and Derek Sleeman. Applying Rule Extraction & Rule Refinement Techniques to (Blackbox) Classifiers. In *Proceedings of the 8th international conference on knowledge capture*, pages 1–5, 2015.

[13] Urbi Chatterjee, Vidya Govindan, Rajat Sadhukhan, Debdeep Mukhopadhyay, Rajat Subhra Chakraborty, Debashis Mahata, and Mukesh 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*, 16(3):424–437, 2019.

[14] Yijun Cui, Chongyan Gu, Qingqing Ma, Yue Fang, Chenghua Wang, Máire O'Neill, and Weiqiang Liu. Lightweight Modeling Attack-Resistant Multiplexer-Based Multi-PUF (MMPUF) Design on FPGA. *Electronics*, 9(5), 2020.

[15] Abhishek Das and Nur A. Touba. A Single Error Correcting Code with One-Step Group Partitioned Decoding Based on Shared Majority-Vote. *Electronics*, 9(5), 2020.

[16] Jeroen Delvaux. Machine-Learning Attacks on PolyPUFs, OB-PUFs, RPUFs, LHS-PUFs, and PUF-FSMs. *IEEE Transactions on Information Forensics and Security*, 14(8):2043–2058, 2019.

[17] Jeroen Delvaux and Ingrid Verbauwhede. Attacking PUF-Based Pattern Matching Key Generators via Helper Data Manipulation. In *Topics in Cryptology – CT-RSA 2014*, volume 8366 of *Lecture Notes in Computer Science*, pages 106–131, Cham, 2014. Springer International Publishing.

[18] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008. Updated by RFCs 5746, 5878, 6176.

[19] D. Dolev and A. Yao. On The Security of Public Key Protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[20] Elena Dubrova, Oscar Näslund, Bernhard Degen, Anders Gawell, and Yang Yu. CRC-PUF: A Machine Learning Attack Resistant Lightweight PUF Construction. In *2019 IEEE European Symposium on Security and Privacy Workshops (EuroSPW)*, pages 264–271, 2019.

[21] Mohammad Ebrahimabadi, Mohamed Younis, and Naghmeh Karimi. A PUF-Based Modeling-Attack Resilient Authentication Protocol for IoT Devices. *IEEE Internet of Things Journal*, 9(5):3684–3703, 2022.

[22] Matthieu Finiasz. Syndrome Based Collision Resistant Hashing. In Johannes Buchmann and Jintai Ding, editors, *Post-Quantum Cryptography*, pages 137–147, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[23] Fatemeh Ganji, Shahin Tajik, Fabian Faessler, and Jean-Pierre Seifert. Strong Machine Learning Attack Against PUFs with No Mathematical Model. volume 9813, 08 2016.

[24] Yansong Gao, Hua Ma, Said F. Al-Sarawi, Derek Abbott, and Damith C. Ranasinghe. PUF-FSM: A Controlled Strong PUF. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(5):1104–1108, 2018.

[25] Yansong Gao, Yang Su, Wei Yang, Shiping Chen, Surya Nepal, and Damith C. Ranasinghe. Building Secure SRAM PUF Key Generators on Resource Constrained Devices. In *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 912–917, 2019.

[26] M.W Gardner and S.R Dorling. Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric Environment*, 32(14):2627–2636, 1998.

[27] Blaise Gassend, Dwaine Clarke, Marten van Dijk, and Srinivas Devadas. Silicon Physical Random Functions. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS '02, page 148–160, New York, NY, USA, 2002. Association for Computing Machinery.

[28] Blaise Gassend, Marten Van Dijk, Dwaine Clarke, Emina Torlak, Srinivas Devadas, and Pim Tuyls. Controlled Physical Random Functions and Applications. *ACM Trans. Inf. Syst. Secur.*, 10(4), jan 2008.

[29] Blaise Gassend, Daihyun Lim, Dwaine Clarke, Marten van Dijk, and Srinivas Devadas. Identification and Authentication of Integrated Circuits. *Concurrency and Computation: Practice and Experience*, 16(11):1077–1098, 2004.

[30] Prosanta Gope, Jemin Lee, and Tony Q. S Quek. Lightweight and Practical Anonymous Authentication Protocol for RFID Systems Using Physically Unclonable Functions. *IEEE Transactions on Information Forensics and Security*, 13(11):2831–2843, 2018.

[31] Prosanta Gope, Owen Millwood, and Biplab Sikdar. A Scalable Protocol Level Approach to Prevent Machine Learning Attacks on Physically Unclonable Function Based Authentication Mechanisms for Internet of Medical Things. *IEEE Transactions on Industrial Informatics*, 18(3):1971–1980, 2022.

[32] Margherita Grandini, Enrico Bagli, and Giorgio Visani. Metrics for Multi-Class Classification: an Overview, 2020.

[33] Chongyan Gu, Chip-Hong Chang, Weiqiang Liu, Shichao Yu, Yale Wang, and Maire O'Neill. A Modeling Attack Resistant Deception Technique for Securing Lightweight-PUF-Based Authentication. *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems*, 40(6):1183–1196, 2021.

[34] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: cold-boot attacks on encryption keys. *Commun. ACM*, 52(5):91–98, may 2009.

[35] Chris Hawkins. *A History of Signatures: From Cave Paintings to Robo-Signings*. Createspace Independent Publishing Platform, 2011.

[36] Zhangqing He, Wanbo Chen, Lingchao Zhang, Gaojun Chi, Qi Gao, and Lein Harn. A Highly Reliable Arbiter PUF with Improved Uniqueness in FPGA Implementation using bit-self-test. *IEEE access*, 8:181751–181762, 2020.

[37] Matthias Hiller, Ludwig Kürzinger, and Georg Sigl. Review of Error Correction for PUFs and Evaluation on State-of-the-art FPGAs. *Journal of cryptographic engineering*, 10(3):229–247, 2020.

[38] Matthias Hiller, Meng-Day (Mandel) Yu, and Michael Pehl. Systematic Low Leakage Coding for Physical Unclonable Functions. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, ASIA CCS '15, page 155–166, New York, NY, USA, 2015. Association for Computing Machinery.

[39] Daniel E. Holcomb, Wayne P. Burleson, and Kevin Fu. Power-Up SRAM State as an Identifying Fingerprint and Source of True Random Numbers. *IEEE Transactions on Computers*, 58(9):1198–1210, 2009.

[40] 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 *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2740–2743, 2014.

[41] Jeremie S Kim, Minesh Patel, Hasan Hassan, and Onur 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)*, volume 2018-, pages 194–207. IEEE, 2018.

[42] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping bits in memory without accessing them: An experimental study of dram disturbance errors. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pages 361–372, 2014.

[43] Vlastimil Klima and Tomas Rosa. Side Channel Attacks on CBC Encrypted Messages in the PKCS7 Format. Cryptology ePrint Archive, Paper 2003/098, 2003. `https://eprint.iacr.org/2003/098`.

[44] Raghavan Kumar and Wayne Burleson. Hybrid Modeling Attacks on Current-based PUFs. In *2014 IEEE 32nd International Conference on Computer Design (ICCD)*, pages 493–496, 2014.

[45] Raghavan Kumar and Wayne Burleson. On Design of a Highly Secure PUF based on Non-Linear Current Mirrors. In *2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 38–43, 2014.

[46] Sachin Kumar, Prayag Tiwari, and Mikhail Zymbler. Internet of Things is a Revolutionary Approach for Future Technology Enhancement: A Review. *Journal of Big Data*, 6, 12 2019.

[47] Lieneke Kusters and Frans M. J Willems. Secret-Key Capacity Regions for Multiple Enrollments With an SRAM-PUF. *IEEE Transactions on Information Forensics and Security*, 14(9):2276–2287, 2019.

[48] Hung-Peng Lee, Shao-I Chu, and Hsin-Chiu Chang. Efficient Decoding of the (23, 12, 7) Golay Code up to Five Errors. *Information Sciences*, 253:170–178, 2013.

[49] Jae W. Lee, Daihyun Lim, Blaise Gassend, G. Edward Suh, Marten van Dijk, and Srinivas Devadas. A Technique to Build a Secret Key in Integrated Circuits for Identification and Authentication Applications. In *2004 Symposium on VLSI Circuits. Digest of Technical Papers (IEEE Cat. No.04CH37525)*, pages 176–179, 2004.

[50] Junhua Li, Zbigniew Struzik, Liqing Zhang, and Andrzej Cichocki. Feature Learning from Incomplete EEG with Denoising Autoencoder. *Neurocomputing (Amsterdam)*, 165:23–31, 2015.

[51] Ruizhe Li, Xiao Li, Guanyi Chen, and Chenghua Lin. Improving Variational Autoencoder for Text Modelling with Timestep-Wise Regularisation. In *Proceedings of the 28th International Conference on Computational Linguistics (COLING)*, 2020.

[52] Jamie Liu, Ben Jaiyen, Yoongu Kim, Chris Wilkerson, and Onur Mutlu. An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms. *SIGARCH Comput. Archit. News*, 41(3):60–71, June 2013.

[53] Jiahao Liu, Yuanzhe Zhao, Yan Zhu, Chi-Hang Chan, and Rui Paulo Martins. A Weak PUF-Assisted Strong PUF With Inherent I]mmunity to Modeling Attacks and Ultra-Low BER, year=2022, volume=69, number=12, pages=4898-4907, doi=10.1109/TCSI.2022.3206214. *IEEE Transactions on Circuits and Systems I: Regular Papers*.

[54] Wenchao Liu, Zhenhua Zhang, Miaoxin Li, and Zhenglin Liu. A Trustworthy Key Generation Prototype Based on DDR3 PUF for Wireless Sensor Networks. In *2014 International Symposium on Computer, Consumer and Control*, pages 706–709, 2014.

[55] Ting Lu, Ryan Kenny, and Sean Atsatt. Secure Device Manager for Intel ®Stratix ®10 Devices Provides FPGA and SoC Security, *Intel*. `https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/wp/wp-01252-secure-device-manager-for-fpga-soc-security.pdf`.

[56] Qingqing Ma, Chongyan Gu, Neil Hanley, Chenghua Wang, Weiqiang Liu, and Maire O'Neill. A Machine Learning Attack Resistant Multi-PUF Design on FPGA. In *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 97–104. IEEE, 2018.

[57] Roel Maes. *Physically Unclonable Functions: Constructions, Properties and Applications*, volume 9783642413957. 11 2013.

[58] Abhranil Maiti and Patrick Schaumont. A Novel Microprocessor-Intrinsic Physical Unclonable Function. In *22nd International Conference on Field Programmable Logic and Applications (FPL)*, pages 380–387, 2012.

[59] Nathan Menhorn. External Secure Storage Using the PUF, *Xilinx*. `https://www.xilinx.com/support/documentation/application_notes/xapp1333-external-storage-puf.pdf`, Jun 2018.

[60] Owen Millwood, Prosanta Gope, Chenghua Lin, Elif Bilge Kavun, Jack Miskelly, and Bohao Yang. DRAM Latency-PUF Responses and corresponding PUF Phenotype conversions: Temperature and Voltage environmentally tested. `https://orda.shef.ac.uk/articles/dataset/DRAM_Latency-PUF_Responses_and_corresponding_PUF_Phenotype_conversions_Temperature_and_Voltage_environmentally_tested/26977528"`, 9 2024.

[61] Owen Millwood, Prosanta Gope, Bohao Yang, Elif Bilge Kavun, Chenghua Lin, and Jack Miskelly. PUF Phenotype CNN-based Authentication Experimental Code and ML Models. `https://orda.shef.ac.uk/articles/software/PUF_Phenotype_CNN-based_Authentication_Experimental_Code_and_ML_Models/27094222",doi="10.15131/shef.data.27094222.v1`, 9 2024.

[62] Owen Millwood, Fei Hongming, Prosanta Gope, Meltem Kurt Pehlivanoglu, Elif Kavun, and Biplab Sikdar. Semi-Honest Verifier Generic PUF Obfuscation Scheme ML CRP Dataset. `https://orda.shef.ac.uk/articles/dataset/Semi-Honest_Verifier_Generic_PUF_Obfuscation_Scheme_ML_CRP_Dataset/26977237`, 9 2024.

[63] Owen Millwood, Jack Miskelly, Bohao Yang, Prosanta Gope, Elif Bilge Kavun, and Chenghua Lin. PUF-Phenotype: A Robust and Noise-Resilient Approach to Aid Group-based Authentication with DRAM-PUFs Using Machine Learning. *IEEE Transactions on Information Forensics and Security*, pages 1–1, 2023.

[64] Owen Millwood, Meltem Kurt Pehlivanoglu, Aryan Mohammadi Pasikhani, Prosanta Gope, Jack Miskelly, and Elif Bilge Kavun. Simulated 16 32 stage Normal, Feed-Forward and XOR Arbiter PUF CRPs. `https://orda.shef.ac.uk/articles/dataset/Simulated_16_32_stage_Normal_Feed-Forward_and_XOR_Arbiter_PUF_CRPs/26977279`, 9 2024.

[65] Owen Millwood, Meltem Kurt Pehlivanoğlu, Elif Bilge Kavun, Prosanta Gope, Jack Miskelly, and Aryan Mohammadi Pasikhani. Generic PUF Obfuscation Scheme HDL Designs and ML Attack Experimental Source Code. `https://orda.shef.ac.uk/articles/software/Generic_PUF_Obfuscation_Scheme_HDL_Designs_and_ML_Attack_Experimental_Source_Code/27095215`, 9 2024.

[66] Owen Millwood, Meltem Kurt Pehlivanoğlu, Elif Bilge Kavun, Biplab Sikdar, Fei Hongming, and Prosanta Gope. Semi-Honest Verifier Generic

PUF Obfuscation Scheme HDL Designs and ML_MA Experimental Code. `https://orda.shef.ac.uk/articles/software/Semi-Honest_Verifier_Generic_PUF_Obfuscation_Scheme_HDL_Designs_and_ML_MA_Experimental_Code/27095617`, 9 2024.

[67] Jack Miskelly, Chongyan Gu, Qingqing Ma, Yijun Cui, Weiqiang Liu, and Máire O'Neill. Modelling attack analysis of configurable ring oscillator (cro) puf designs. In *2018 IEEE 23rd international conference on digital signal processing (DSP)*, pages 1–5. IEEE, 2018.

[68] Jack Miskelly and Maire O'Neill. Fast DRAM PUFs on Commodity Devices. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39:1–1, 11 2020.

[69] Sergey Morozov, Abhranil Maiti, and Patrick Schaumont. A comparative analysis of delay based puf implementations on fpga. Cryptology ePrint Archive, Paper 2009/629, 2009. `https://eprint.iacr.org/2009/629`.

[70] Fatemeh Najafi, Masoud Kaveh, Diego Martín, and Mohammad Reza Mosavi. Deep PUF: A Highly Reliable DRAM PUF-Based Authentication for IoT Networks Using Deep Convolutional Neural Networks. *MDPI Sensors*, 21(6), 2021.

[71] Ravikanth Srinivasa Pappu. *Physical One-Way Functions*. PhD thesis, Massachusetts Institute of Technology, 2001.

[72] Jan M. Rabaey, Anantha Chandrakasan, and Borivoje Nikolic. *Digital Integrated Circuits- A Design Perspective*. Prentice Hall, 2ed edition, 2004.

[73] Sudipta Roy, Mahtab Ahmed, and M. A. H. Akhand. Noisy Image Classification Using Hybrid Deep Learning Methods. *Journal of Information and Communication Technology*, 17:233–269, 04 2018.

[74] Ulrich Rührmair, Frank Sehnke, Jan Sölter, Gideon Dror, Srinivas Devadas, and Jürgen Schmidhuber. Modeling Attacks on Physical Unclonable Functions. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, page 237–249, New York, NY, USA, 2010. Association for Computing Machinery.

[75] Ulrich Rührmair and Marten van Dijk. PUFs in Security Protocols: Attack Models and Security Evaluations. In *2013 IEEE symposium on security and privacy*, pages 286–300. IEEE, 2013.

[76] Ulrich Rührmair, Jan Sölter, Frank Sehnke, Xiaolin Xu, Ahmed Mahmoud, Vera Stoyanova, Gideon Dror, Jürgen Schmidhuber, Wayne Burleson, and Srinivas Devadas. PUF Modeling Attacks on Simulated and Silicon Data. *IEEE Transactions on Information Forensics and Security*, 8(11):1876–1891, 2013.

[77] Pranesh Santikellur, Aritra Bhattacharyay, and Rajat Subhra Chakraborty. Deep Learning based Model Building Attacks on Arbiter PUF compositions. *IACR Cryptol. ePrint Arch.*, page 566, 2019.

[78] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition, 2014.

[79] Emanuele Strieder, Christoph Frisch, and Michael Pehl. Machine Learning of Physical Unclonable Functions using Helper Data: Revealing a Pitfall in the Fuzzy Commitment Scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(2):1–36, Feb. 2021.

[80] G. Edward Suh and Srinivas Devadas. Physical Unclonable Functions for Device Authentication and Secret Key Generation. In *2007 44th ACM/IEEE Design Automation Conference*, pages 9–14, 2007.

[81] B. M. S. Bahar Talukder, Biswajit Ray, Domenic Forte, and Md. Tauhidur. Rahman. PreLatPUF: Exploiting DRAM Latency Variations for Generating Robust Device Signatures. *IEEE Access*, 7:81106–81120, 2019.

[82] Peter J. Taylor and Richard C. Lewontin. The Genotype/Phenotype Distinction. 2017.

[83] Fatemeh Tehranipoor, Nima Karimian, Wei Yan, and John A. Chandy. DRAM-Based Intrinsic Physically Unclonable Functions for System-Level Security and Authentication. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(3):1085–1097, 2017.

[84] Fatemeh Tehranipoor, Nima Karimina, Kan Xiao, and John Chandy. Dram based intrinsic physical unclonable functions for system level security. 05 2015.

[85] Ioannis Tsiokanos, Jack Miskelly, Chongyan Gu, Maire O'neill, and Georgios Karakonstantis. DTA-PUF: Dynamic Timing-Aware Physical Unclonable Function for Resource-Constrained Devices. *J. Emerg. Technol. Comput. Syst.*, 17(3), aug 2021.

[86] Vincent van der Leest, Erik van der Sluis, Geert-Jan Schrijen, Pim Tuyls, and Helena Handschuh. Efficient Implementation of True Random Number Generator Based on SRAM PUFs. In David Naccache, editor, *Cryptography and Security: From Theory to Applications: Essays Dedicated to Jean-Jacques Quisquater on the Occasion of His 65th Birthday*, pages 300–318. Springer Berlin Heidelberg, 2012.

[87] Henk CA Van Tilborg and Sushil Jajodia. *Encyclopedia of Cryptography and Security*. Springer Science & Business Media, 2014.

[88] Ingrid Verbauwhede and Roel Maes. Physically unclonable functions: manufacturing variability as an unclonable device identifier. In *Proceedings of the 21st Edition of the Great Lakes Symposium on Great Lakes Symposium on VLSI*, GLSVLSI '11, page 455–460, New York, NY, USA, 2011. Association for Computing Machinery.

[89] Nils Wisiol, Christoph Gräbnitz, Christopher Mühl, Benjamin Zengin, Tudor Soroceanu, Niklas Pirnay, Khalid T. Mursi, and Adomas Baliuka. PyPuf: Cryptanalysis of Physically Unclonable Functions, 2021.

[90] Nils Wisiol, Christopher Mühl, Niklas Pirnay, Phuong Ha Nguyen, Marian Margraf, Jean-Pierre Seifert, Marten van Dijk, and Ulrich Rührmair. Splitting the Interpose PUF:

A Novel Modeling Attack Strategy. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(3):97–120, Jun. 2020.

[91] Wenjie Xiong, Nikolaos Athanasios Anagnostopoulos, André Schaller, Stefan Katzenbeisser, and Jakub Szefer. Spying on temperature using dram. In *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 13–18, 2019.

[92] Jing Ye, Yu Hu, and Xiaowei Li. RPUF: Physical Unclonable Function with Randomized Challenge to Resist Modeling Attack. In *2016 IEEE Asian Hardware-Oriented Security and Trust (AsianHOST)*, pages 1–6, 2016.

[93] Meng-Day Yu, Matthias Hiller, Jeroen Delvaux, Richard Sowell, Srinivas Devadas, and Ingrid Verbauwhede. A Lockdown Technique to Prevent Machine Learning on PUFs for Lightweight Authentication. *IEEE Transactions on Multi-Scale Computing Systems*, 2(3):146–159, 2016.

[94] Michael Yue, Nima Karimian, Wei Yan, Nikolaos Athanasios Anagnostopoulos, and Fatemeh Tehranipoor. DRAM-Based Authentication Using Deep Convolutional Neural Networks. *IEEE Consumer Electronics Magazine*, 10(4):8–17, 2021.

[95] Jiliang Zhang and Chaoqun Shen. Set-Based Obfuscation for Strong PUFs Against Machine Learning Attacks. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 68(1):288–300, 2021.

[96] Jiliang Zhang, Chaoqun Shen, Zhiyang Guo, Qiang Wu, and Wanli Chang. CT-PUF: Configurable Tristate PUF Against Machine Learning Attacks for IoT Security. *IEEE Internet of Things Journal*, 9(16):14452–14462, 2022.

[97] Yu Zheng, Fengchao Zhang, and Swarup Bhunia. DScanPUF: A Delay-Based Physical Unclonable Function Built Into Scan Chain. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24(3):1059–1070, 2016.

[98] Baokui Zhu, Xiaowen Jiang, Kai Huang, and Miao Yu. A response-feedback-based strong puf with improved strict avalanche criterion and reliability. *Sensors*, 24(1), 2024.

Appendices

Appendix A

Full VGG16 Feature Extractor Confusion Matrices

We provide the confusion matrices for the test performance of each classifier when using the full VGG16 feature extractor. Figures A.1 and A.2 show the actual class and the class predicted by the model for each classifier for each test sample. Each includes the number of samples predicted for a given class alongside the percentage of predictions of each class across all predictions. Given perfect classifier performance, a confusion matrix should show a value of zero for all cells where the actual and predicted label rows and columns match. For the classifiers using the full VGG16 feature extractor, while it can be seen most classifications are correct, there are frequently classifications that are made outside of the central set of ideal classifications. For the DT classifier for example, many of each class were predicted incorrectly, showing green in many of the cells where actual and predicted label do not match. Overall, there is no significant apparent correlation to be drawn between the predicted value of certain classes and the actual value as incorrect classifications are mostly arbitrarily distributed around the confusion matrices. In some cases, there are similarities however, for example with the RF, SVM and XGB models, the a three images from device *epsilon* were classified as one of *alpha*, *beta* and *delta* respectively. This result is likely caused by the classifiers drawing similar conclusions from the features extracted from identical images impacted by in-image noise.

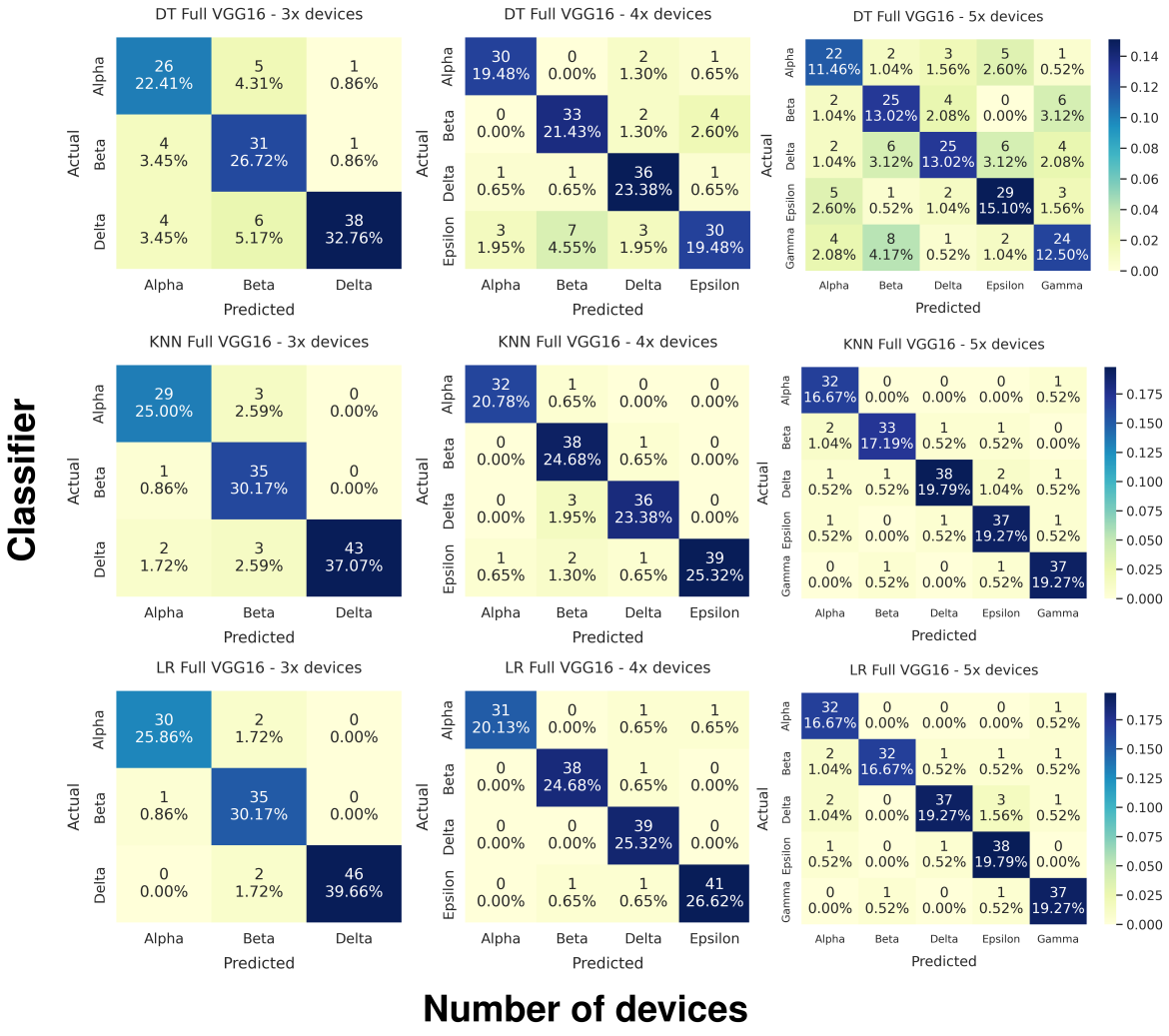

Figure A.1: *Confusion Matrices for DT, KNN & LR Classifiers Using Full VGG16 Feature Extractor*

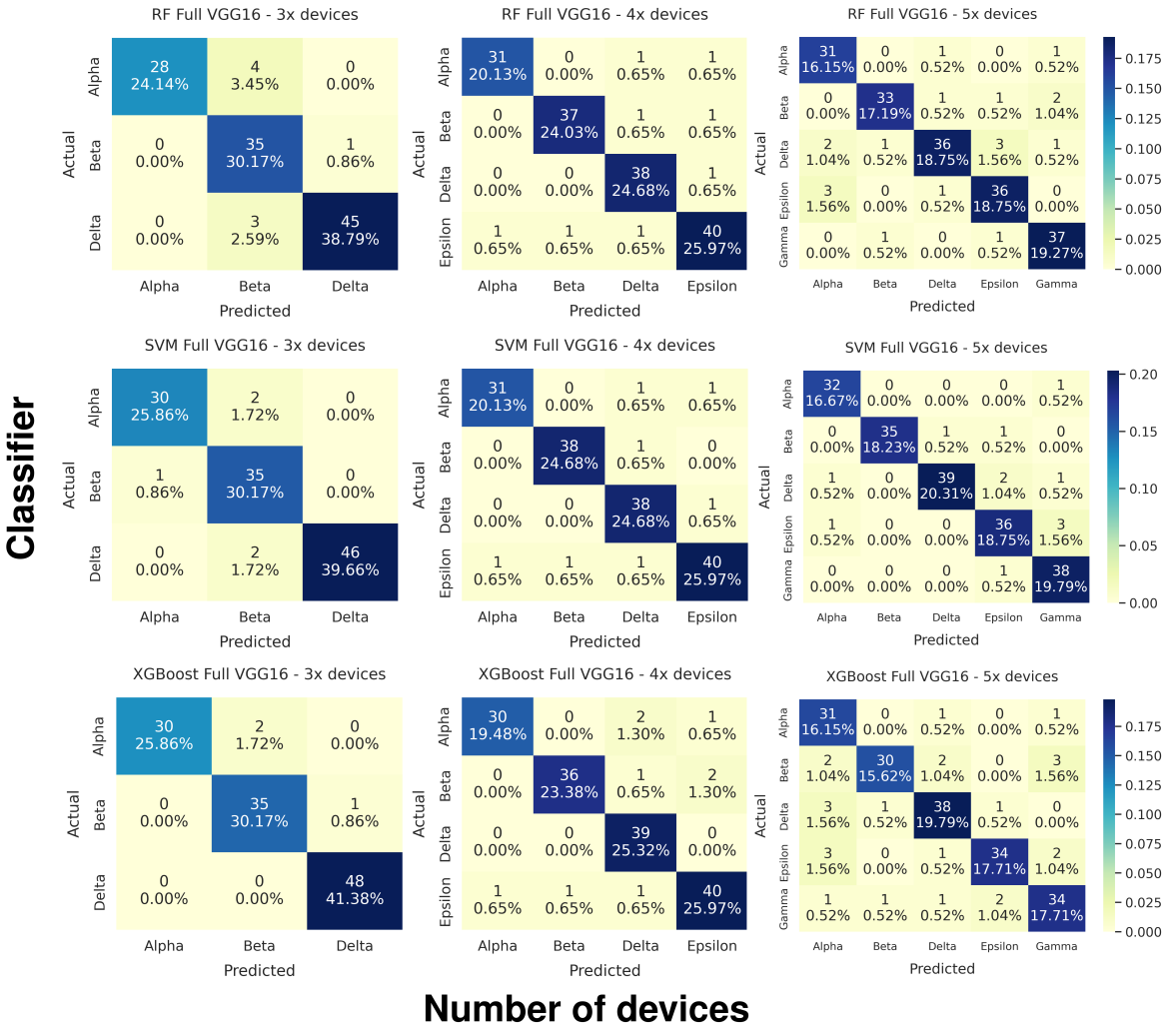

Figure A.2: *Confusion Matrices for RF, SVM & XGB Classifiers Using Full VGG16 Feature Extractor*

Appendix B

Lightweight VGG16 Feature Extractor Confusion Matrices

Here we provide the confusion matrices for each classifier when using the modified lightweight VGG16 feature extractor (excluding RF and SVM as these are provided in Figure 7 of Section IV of the main paper). This lightweight feature extractor has the dense layers of the VGG16 removed, and is the basis of our proposed scheme. As can be seen in each confusion matrix, most of the classifiers performed very strongly in their predictions for each test sample, with only few samples displayed outside of the centre line.

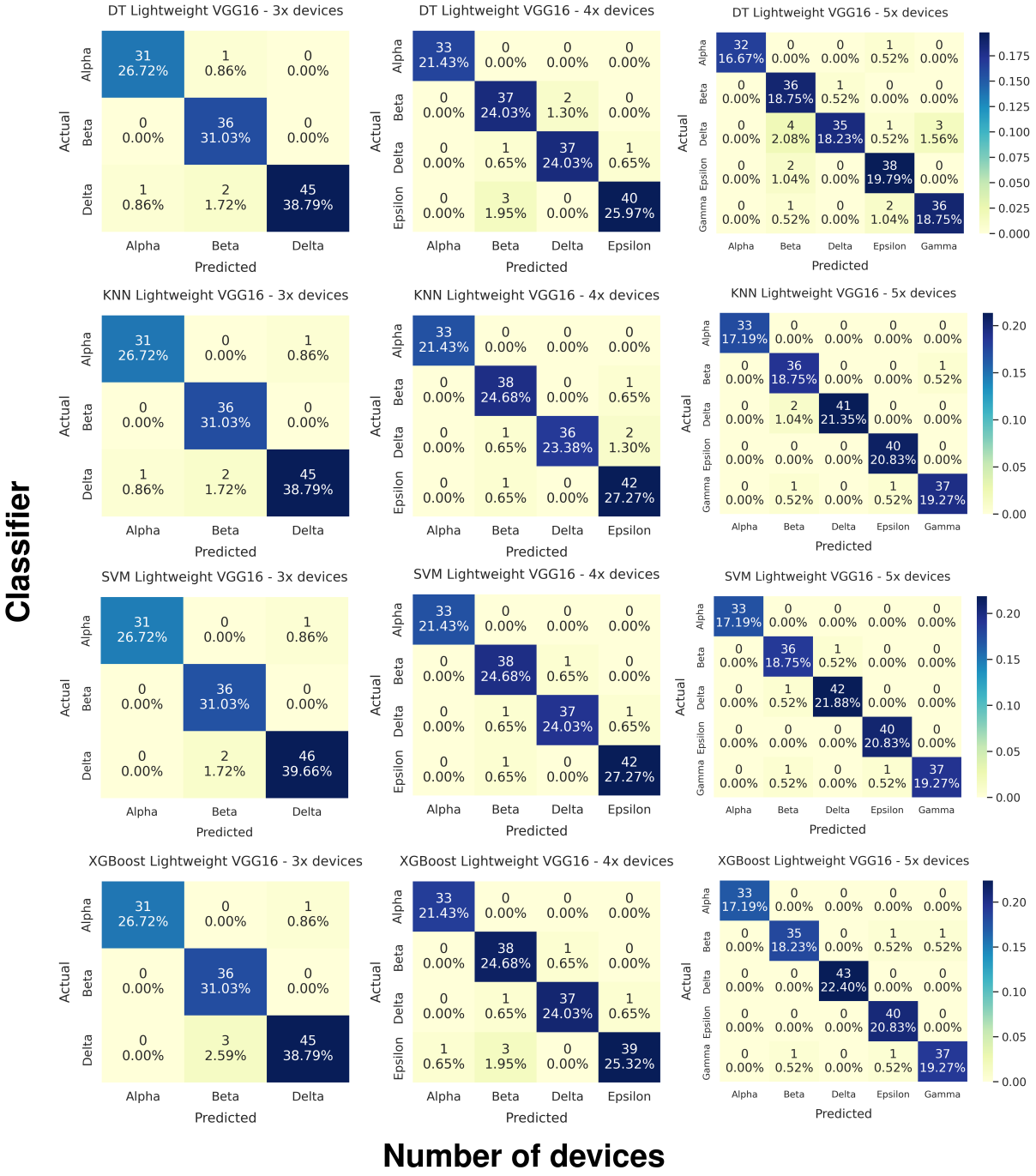

Figure B.1: *Confusion Matrices for DT, KNN, SVM & XGB Classifiers Using Lightweight VGG16 Feature Extractor*

Author's Publications

- **Millwood, O.**, Miskelly, J., Yang, B., Gope, P., Kavun, E. B., and Lin, C. (2023). *PUF-Phenotype: A Robust and Noise-Resilient Approach to Aid Group-Based Authentication With DRAM-PUFs Using Machine Learning.* IEEE Transactions on Information Forensics and Security (TIFS).

- **Millwood, O.**, Pehlivanoglu, M. K., Pasikhani, A. M., Miskelly, J., Gope, P., and Kavun, E. B. (2023). *A Generic Obfuscation Framework for Preventing ML-Attacks on Strong-PUFs through Exploitation of DRAM-PUFs.* 8th IEEE European Symposium on Security and Privacy (EuroS&P).

- **Millwood, O.**, Hongming, F., Gope, P., Narlı, O., Pehlivanoğlu, M. K., Kavun, E. B., and Sikdar, B. (2023). *A Privacy-Preserving Protocol Level Approach to Prevent Machine Learning Modelling Attacks on PUFs in the Presence of Semi-Honest Verifiers.* 2023 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), 326–336. IEEE.

- Gope, P., **Millwood, O.**, and Sikdar, B. (2021). *A Scalable Protocol Level Approach to Prevent Machine Learning Attacks on Physically Unclonable Function Based Authentication Mechanisms for Internet of Medical Things.* IEEE Transactions on Industrial Informatics (TII).

- Gope, P., **Millwood, O.** and Saxena, N. (2021). *A Provably Secure Authentication Scheme for RFID-enabled UAV Applications.* Computer Communications

- Bartsch, W., **Millwood O.** and Kavun, E. B.. (2023) *Zero Knowledge Registration of PKI Authentication for Symbiotic Security in FIDO IoT.* J. Surveill. Secur. Saf., 4, 155-79.

- Bartsch, W., Gope, P., Kavun, E., **Millwood, O.**, Panchenko, A., Pasikhani, A. and Polian, I. (2023). *Design Rationale for Symbiotically Secure Key Management Systems in IoT and Beyond.* In Proceedings of the 9th International Conference on Information Systems Security and Privacy - ICISSP

- Hesse, D., Gay, M. Polian, I., Kavun, E. B., **Millwood, O.**, and Bartsch W. (2023) *A Modular Open-Source Cryptographic Co-Processor for Internet of Things.* In Proceedings of the 26th Euromicro Conference on Digital System Design (AHSA: Architectures and Hardware for Security Applications)