Methodologies for the Design, the Modeling, and the Quality Assessment of Physical Unclonable Functions (PUFs)

Sergio Vinagrero Gutierrez

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

École doctorale : EEATS - Electronique, Electrotechnique, Automatique, Traitement du Signal (EEATS)
Spécialité : Nano électronique et Nano technologies
Unité de recherche : Techniques de l'Informatique et de la Microélectronique pour l'Architecture des systèmes intégrés

Méthodologies pour la Conception, la Modélisation et l'Évaluation de la Qualité des Fonctions Physiques Non Clonables (PUFs)

Methodologies for the Design, the Modeling, and the Quality Assessment of Physical Unclonable Functions (PUFs)

Présentée par :

Sergio VINAGRERO GUTIERREZ

Direction de thèse :

Giorgio DI NATALE DIRECTEUR DE RECHERCHE, Université Grenoble Alpes	Directeur de thèse
Elena-Ioana VATAJELU CHARGEE DE RECHERCHE, Université Grenoble Alpes	Co-encadrante de thèse

Rapporteurs :

Sylvain GUILLEY
PROFESSEUR DES UNIVERSITES, Télécom Paris
Jean-Pierre SEIFERT
PROFESSEUR DES UNIVERSITES, Université technique de Berlin

Thèse soutenue publiquement le **27 septembre 2024**, devant le jury composé de :

Giorgio DI NATALE, DIRECTEUR DE RECHERCHE, Université Grenoble Alpes	Directeur de thèse
Sylvain GUILLEY, PROFESSEUR DES UNIVERSITES, Télécom Paris	Rapporteur
Jean-Pierre SEIFERT, PROFESSEUR DES UNIVERSITES, Université technique de Berlin	Rapporteur
Lionel TORRES, PROFESSEUR DES UNIVERSITES, Université de Montpellier	Examinateur
Vincent BEROULLE, PROFESSEUR DES UNIVERSITES, Grenoble INP - UGA	Examinateur
Noemi BERINGUIER-BOHER, DOCTEURE EN SCIENCES, Synopsys	Examinatrice

Invités :

Honorio Martin Gonzalez
ASSOCIATE PROFESSOR, Universidad Carlos III
Ioana Vatajelu
CHARGEE DE RECHERCHE HDR, TIMA Laboratory

Abstract

Physical Unclonable Functions (PUFs) are a promising alternative to conventional cryptographic methods for securing sensitive data in modern circuits by generating unique secrets on the fly, leveraging inherent process variability and eliminating the need for data storage. Ring Oscillator and SRAM-based PUFs are particularly studied due to their simplicity and prevalence in System-on-Chips (SOCs). A major focus was placed on the relationship between entropy and reliability in PUFs, leading to the development of a simulation-based methodology for setting reliability thresholds based on frequency differences. Subsequently, a holistic mathematical model accounting for process variability was created to optimize RO-PUF designs, and a new design methodology, "Split PUF," is introduced to maximize entropy yield. The mathematical modeling of PUFs, a less-explored area is also investigated, by proposing statistical and numerical methodologies to improve understanding of RO and SRAM-based PUF designs. Statistical methods for metric extrapolation are introduced, reducing the time and cost needed to evaluate PUFs. Furthermore, digital twins of PUFs are proposed, facilitating algorithm testing and evaluation. These models provide a robust and cost-effective methodology for assessing PUF performance and aid in their security assessment. During the parametric simulations for PUF evaluation, several limitations in available commercial Electronic Design Automation (EDA) software were identified. To address these challenges, a series of open-source tools were developed, such as Monaco and NIMPHEL, to simplify and accelerate the design process and evaluation through simulation methodologies. To validate the simulation results, an open-source platform, SRAMPlatform, was created to gather extensive SRAM data and sensor readings from microcontrollers. The platform gathers data from 84 STM32 microcontrollers, with weekly updates stored in an open-access database, addressing the scarcity of accessible PUF datasets. Additionally, a comprehensive dataset from Infineon provided valuable insights for validating simulation hypotheses and exploring new PUF designs. Furthermore, significant limitations in standard PUF performance metrics are noted and several mitigations and new alternative metrics for more robust evaluation are proposed. Real-world data from the SRAM platform showed extreme bias and correlation effects that the canonical metrics failed to highlight, underscoring the need for more robust testing methodologies to accurately identify these effects. These advancements enhance PUF assessment methodologies, addressing limitations in current tools and metrics, and providing new frameworks and models for future research. Future work includes developing a unified testing framework for all PUF families, validating statistical models across a wider spectrum of PUF families, refining the digital twin models, extending the concept of Split PUFs to new PUF families. These research directions aim to accelerate the worldwide adoption of PUF technologies by enhancing assessment methodologies, addressing current tool and metric limitations, and providing new frameworks and models for future research.

Contributions

This manuscript is organized in 7 parts. The structure of this manuscript is illustrated below and the specific contributions in each of areas have been highlighted.

The first part introduces the critical need for security in modern systems along with the introduction of Physical Unclonable Functions shortplural (PUFs) as security primitives for cryptographic applications. Various PUFs architectures are introduced and the significant challenges these technologies face are outlined.

The second part describes the simulation environment and workflow utilized throughout the thesis. Several frameworks and software tools were developed and integrated into the simulation workflow. Notably, Monaco (accessible at https://servinagrero.github.io/monaco and published in [1]) and NIMPHEL (available at https://servinagrero.github.io/nimphel and published in [2]). These tools were essential for simulating the real environment and designs, providing a robust platform for testing and validating PUF performance.

Part III focuses on the studied PUF designs and datasets. A platform, coined SRAMPlatform, accessible at https://servinagrero.github.io/SRAMPlatform and published in [3], [4] [5], was developed from scratch to collect extensive Static Random Access Memory (SRAM) data from microcontrollers for reliability testing. The Ring Oscillator PUF simulated is also presented. Additionally, experimental data from Infineon devices including Ring Oscillators shortplural (ROs) are also utilized. This extensive dataset was crucial for corroborating the simulation findings with real-world performance metrics.

Part IV, dedicated to PUF evaluation metrics, provides an in-depth analysis of both canonical and alternative evaluation metrics for PUFs. The limitations of existing metrics are thoroughly examined, include the lack of a golden model, correlation and auto-correlation analysis, and the insufficient study of uniqueness limits. Several mitigations proposals are proposed, including new metrics like Stability, Reliability Invariance, and Punctual Bit-aliasing, along with the foundation for a comprehensive test suite, akin to a ”NIST test suite for PUFs,” tailored specifically for evaluating PUFs.

In part V, the intricate relationship between Reliability and Entropy in PUFs is investigated. A ”Time-to-response” model is introduced to correlate RO frequency differences with response reliability. This research revealed a previously underexplored relationship between frequency difference and entropy. A holistic RO model is developed to encapsulate these effects, facilitating the quantification and modeling of RO PUFs. The theoretical findings were validated with simulation data, and real-world data further confirmed the interconnection between reliability and entropy through frequency differences. The statistical analysis libraries created in an effort of open standardization, are available at https://servinagrero.github.io/pufr, with findings published in [6], [7] and [8].

Building on the holistic model from the previous section, part VI introduces the ”Split PUF”, an enhanced version of the RO-PUF, which can limit the lack of entropy with an easy design, suitable for chiplet-based designs. The holistic model developed allows this

enhancement to potentially be applied to other delay-based or various PUF families in the future.

As a novel work, the knowledge gained from previous parts is employed to develop robust mathematical models for the creation of PUF digital twins. These digital twins can predict and extrapolate the results of canonical metrics from small test datasets. Indeed, software-based models are created capable of replicating the behaviour and unique characteristics of both RO and SRAM PUFs. These models facilitate the easy evaluation and assessment of different PUF designs, offering a powerful tool for future research and assessment to accelerate their world-wide adoption.

Finally, a general summary of the document is provided, synthesizing the key findings and contributions from each section. I highlight the advancements made in understanding, evaluating, and enhancing PUFs. Additionally, I discuss perspectives for future research, emphasizing areas that require further refinement. This includes the need for more comprehensive analysis of PUF evaluation metrics, the exploration of new PUF designs leveraging the developed holistic models, and the continuous development of standardized test suites to ensure consistent and reliable benchmarking of PUF technologies. Through these contributions, this thesis advances the understanding, evaluation, and enhancement of PUFs, paving the way for more secure and reliable cryptographic applications.

Publications

Conferences

[1] S. V. Gutierrez, G. Di Natale, and E.-I. Vatajelu, 'On-Line Method to Limit Unreliability and Bit-Aliasing in RO-PUF,' in *2023 IEEE 29th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, IEEE, 2023, pp. 1–6.

[2] S. Vinagrero Gutierrez, G. Di Natale, and I. Vatajelu, 'On-Line Reliability Estimation of Ring Oscillator PUF,' in *IEEE European Test Symposium (ETS 2022)*, IEEE, Ed., Barcelona, Spain: IEEE, May 2022. DOI: 10.1109/ETS54262.2022.9810418. [Online]. Available: https://hal.archives-ouvertes.fr/hal-03767650.

[3] S. V. Gutiérrez, P. Inglese, G. Di Natale, and E.-I. Vatajelu, 'Open automation framework for complex parametric electrical simulations,' in *2023 26th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, 2023, pp. 132–135. DOI: 10.1109/DDECS57882.2023.10139409.

Journals

[1] S. Vinagrero Gutiérrez, G. Di Natale, and E.-I. Vatajelu, 'Python framework for modular and parametric spice netlists generation,' *Electronics*, vol. 12, no. 18, p. 3970, 2023.

[2] S. Vinagrero, H. Martin, A. de Bignicourt, E.-I. Vatajelu, and G. Di Natale, 'SRAM-Based PUF Readouts,' *Scientific Data*, vol. 10, no. 1, p. 333, 2023, ISSN: 2052-4463. DOI: 10.1038/s41597-023-02225-9.

[3] V. Kulagin *et al.*, 'On the Relation Between Reliability and Entropy in Physical Unclonable Functions,' *IEEE Design & Test*, pp. 1–1, 2024. DOI: 10.1109/MDAT.2024.3425791.

Journals Under submission

[1] S. V. Gutierrez, G. Di Natale, and E.-I. Vatajelu, 'On the Limitations of PUF Canonical Evaluation Metrics,'

[2] S. V. Gutierrez, G. Di Natale, and E.-I. Vatajelu, 'Split PUF Design Methodology,'

[3] S. V. Gutierrez, G. Di Natale, and E.-I. Vatajelu, 'Leak Attack on SRAM-PUFs,'

Open Access tools and data

The tools developed are open source released under the MIT licence and are accessible under github repositories. They are accessible at https://servinagrero.github.io/monaco, https://servinagrero.github.io/NIMPHEL and https://servinagrero.github.io/pufr.

The source code for the SRAMPlatform and the source code for the STM32 devices along with the usage documentation is available under the GPL-2.0 licence at https://servinagrero.github.io/SRAMPlatform.

A static version of the full dataset available at the time of writing can be downloaded from Zenodo [5] at https://zenodo.org/doi/10.5281/zenodo.7529512. The full dataset is composed of 2 CSV files that contain SRAM readouts of 84 Nucleo microcontrollers of STM32 type along with their voltage and temperature sensor data. Although the dataset described in this document is static, more data can be requested online through the following application hosted at https://puf4iot.univ-grenoble-alpes.fr/form.php. Any user can submit a request to the server specifying the data they want to retrieve. The server will process the request and return the data in a CSV file or in a zip file if the number of records is too large.

Table of contents

List of Figures

List of Tables

List of Listings

Glossary

Challenge	A challenge is defined as a numeric index to select a unique response from a PUF
Enrollment	Process of obtaining the first sample of responses from a PUF to store it in the CRP database
Interrogation	Process of providing one or multiple challenges and obtaining various response bits (strong PUF) or the whole response (weak PUF)
Repeatability	Used as synonym of Reliability, rpresents the ability of a system to produce the same output given the same inputs.
Response	Output of the PUF after the interrogation process
SPICE	Simulation Program with Integrated Circuit Emphasis
Verification	Process of applying a challenge from the CRP database to a PUF and check its response

Acronyms

AMS	Analog and Mixed Signal
ASIC	Application-Specific Integrated Circuit
CAD	Computer Aided Design
CDF	Cumulative Distribution Function
CI	Confidence Interval
CMOS	Complementary Metal Oxide Semiconductor
CRP	Challenge Response Pair
DSE	Design Space Exploration
ECC	Error Correcting Code
EDA	Electronic Design Automation
FPGA	Field Programmable Gate Array
IoT	Internet of Things
LUT	Look Up Table
MC	Markov Chain
ML	Machine Learning
MLE	Maximum Likelihood Estimation
NBTI	Negative Bias Temperature Instability
NVM	Non-Volatile Memory
PDF	Probability Density Function
PDK	Process Development Kit
PUF	Physical Unclonable Function
PVTA	Process, Voltage, Temperature and Aging
RNG	Random Number Generator
RO	Ring Oscillator
SNM	Static Noise Margin
SoC	System On Chip

SRAM Static Random Access Memory

SVM Support Vector Machines

I

Introduction

I.1 Introduction

Security in the real world, particularly in the realm of electronics, is of paramount importance as our reliance on digital systems and devices continues to grow. With the proliferation of Internet of Things (IoT) devices, embedded systems, and advanced hardware platforms, ensuring robust security measures is crucial to protect sensitive data, prevent unauthorized access, and maintain the integrity of critical infrastructure. In the field of electronics, security threats manifest in numerous ways, including hardware hacks, side-channel attacks, and malicious software intrusions. These vulnerabilities can lead to significant financial losses, compromise personal privacy, and significant breaches, compromising sensitive information and potentially causing catastrophic failures in critical systems. According to the European Union Agency for Cybersecurity (ENISA), threats like Supply chain compromise of software dependencies, Human error and exploited legacy systems within cyber-physical ecosystems, Targeted attacks enhanced by smart devices data, Rise of advanced hybrid threats are among the leading threats in their forecast for 2030. Additionally, ransomware attacks on critical infrastructure, like the 2021 Colonial Pipeline incident, demonstrate how electronic vulnerabilities can lead to substantial economic and operational disruptions.

Forgery [9] is a real threat of goods nowadays, and it's not only in the context of electronics. It can lead to huge economic losses, distrust between parties and most importantly, hazards to human safety.

Other problems that diminish trust between parties are hardware trojans, [10, 11, 12], which are unauthorized modifications of integrated circuits in order to execute a malicious action triggered by a determined payload.

Side-channel attacks for Systems On Chip shortplural (SoCs) are an active field of research as highlighted in [13, 14]. These type of attacks try to gather information about the behaviour of the circuit either by passive or active means, both being very successful.

To combat these issues traditional security system have employed security primitives to construct secure communication protocols and protect data integrity. These primitives include basic algorithms and protocols such as encryption, hashing, digital signatures, and key exchange mechanisms. Cryptographic systems have long relied on storing sensitive information, such as cryptographic keys and authentication data, in Non-Volatile Memory (NVM) like EEPROM, Flash memory, and hard drives. While these methods offer the advantage of data persistence even when the power is off, they also present significant vulnerabilities. NVMs can be physically tampered with, allowing attackers to extract, modify, or forge critical secrets. Techniques such as probing, reverse engineering, and side-channel attacks enable malicious entities to access the stored data without authorization.

These vulnerabilities highlight the inherent limitations of NVM-based security systems. Since the data remains intact even when the device is off, it is exposed to various physical and electronic attacks that can compromise the entire security framework. Furthermore, once an attacker gains access to the NVM, they can not only read the stored secrets but also alter them, potentially inserting malicious code or forging authentication credentials. These limitations have prompted the development of alternative security approaches that do not rely on static storage of sensitive information.

One such approach involves using PUFs, which leverage the inherent physical variations in semiconductor devices to generate unique and unpredictable responses, making them difficult to duplicate or predict even given complete information about the device. Akin to how no 2 humans have the same fingerprints, there are no 2 circuits alike, even if they are designed equally and manufactured in the same environment. Unlike NVMs, PUFs do not store secrets in a static form; instead, they generate them dynamically in response to specific challenges. This makes PUFs highly resistant to physical attacks and tampering, which makes them ideal for anti-counterfeiting, key distribution, and key storage as already argued by many [15, 16]. By addressing the vulnerabilities associated with NVMs, PUFs and other innovative security primitives offer a promising path forward in the quest to protect sensitive information in electronic systems.

They take their name from the mathematical idea of one-way function [17], that is a mechanism that it's easy to compute for every input, but given the output is very complex to derive the input. When a specific input, known as a challenge, is applied to a PUF, it generates a corresponding output, called a response. The set of challenges and responses of a PUF is known as Challenge-Response Pairs (CRPs). The uniqueness of CRPs is derived from the unique physical characteristics of each PUF, ensuring that the same challenge applied to different PUFs yields different responses, and the same PUF consistently produces the same response to the same challenge under stable conditions. Because the set of CRPs is unique to each device and inherently tied to its physical properties, PUFs are ideal for device authentication and cryptographic key generation. By storing a predefined set of CRPs for a device, authentication involves verifying that the device produces the correct responses to specific challenges. This method guarantees the device's authenticity and ensures it has not been compromised. Moreover, PUFs can dynamically generate cryptographic keys unique to each device. By utilizing specific challenges and their corresponding responses, secure keys can be generated instantly without the need for storing them in non-volatile memory.

There have been many proposals but the first technique to assign a unique identification to every single instance of an IC was proposed in [18]. In the next section, the development of the different PUF designs and entropy sources will be explored in detail.

In hardware security modules, companies like Synopsys through the recently acquired Intrinsic ID, use PUF technology to generate unique cryptographic keys, ensuring robust authentication and protecting sensitive data. Similarly, in the IoT sector, Infineon Technologies, through its acquisition of Cypress Semiconductor, leverages PUFs to secureIoT

devices. These unique identifiers make it difficult for unauthorized entities to access or tamper with the devices.

PUFs also play a significant role in anti-counterfeiting efforts and Field Programmable Gate Array (FPGA)-based system security. Verayo employs PUF-based technologies to create unique fingerprints for products, helping to verify authenticity and combat counterfeiting in industries like pharmaceuticals and luxury goods. Additionally, Xilinx integrates PUFs into FPGAs to protect configuration data and intellectual property, preventing unauthorized access and replication. These applications highlight the versatility and effectiveness of PUFs in enhancing security across diverse domains.

Overview of PUF Architectures

I.2

Although fingerprint identification of humans dates back at least to the 19th century [19], early forms of PUFs were inspired by anti-counterfeit measures used in currency bills [20, 21] almost at the end of the 20th century. The unique and random fibre structures in paper bills were exploited to verify authenticity, leading to the development of Reflective Particle Tags for unclonable identification [20, 22, 23, 24]. However, it wasn't until the early 2000s that PUF technology began to take its modern form. In 2002, Pappu introduced the first optical Physical One Way Function (POWF) [17], and Gassend introduced delay-based PUFs, including the Ring Oscillator PUF, marking the beginning of a new era in hardware security [25].

The successive years saw rapid advancements in PUF technology. Gassend's introduction of the Arbiter PUF in 2003 [26] and Lee's proposal for the Arbiter PUF in 2004 [27] highlighted the additive nature of delay chains, leading to the creation of model-building attacks [25, 27]. To counteract these attacks, Lim developed the Feed-forward Arbiter PUF [28], which introduced intermediate arbiters to create non-linearities in the delay lines. Despite these efforts, arbiter PUFs remained vulnerable to state-of-the-art machine learning attacks [29, 30]. In 2005, Vrijaldenhoven introduced the first acoustical PUF [31], followed by Tuyls' Coating PUF in 2006 [32], which incorporated random elements through a dielectric coating, enhancing security against physical attacks [33].

The period between 2006 and 2009 saw the diversification of PUF designs and applications. Simpson's work in 2006 idealized the use of PUFs in Field-Programmable Gate Arrays (FPGAs) [34], leading to the first FPGA-based SRAM-PUF by Guajardo in 2007 [35]. Dejean proposed the first RF-DNA PUF, which utilized near-field scattering of electromagnetic waves [36]. In 2008, Helinski introduced a PUF based on resistance variations in the power grid of a chip [37], and Oeztuerk adapted a Hopper-Blum style protocol for modelable arbiter PUFs [38]. Holcomb's SRAM-PUF in 2009 provided a robust method for device identification [39], and Guajardo's LC PUF combined capacitive and inductive components, offering a unique approach to PUF design [15].

The advancements continued with the development of various PUF types, including the TERO-PUF proposed by Bossuet in 2013, which improved upon the standard Ring Oscillator PUF [40]. These innovations underscored the versatility of PUFs in enhancing security across diverse domains, from secure key generation and device authentication to anti-counterfeiting measures. The inherent physical variations in semiconductor devices used by PUFs make them difficult to duplicate or predict, providing a robust solution against physical attacks and tampering.

Throughout these developments, researchers also recognized the trade-offs involved in PUF design. Skoric's work in 2005 and 2006 highlighted the challenges of extracting robust keys from noisy PUFs, emphasizing the need to balance the number of extractable

bits with noise levels [41, 33]. These insights have guided the ongoing evolution of PUF technology, ensuring that it remains a critical component of modern hardware security systems. The unique identification capabilities of PUFs, as demonstrated by Lofstrom's ICID technique [18], continue to drive innovations in secure electronic systems, offering promising solutions for future applications.

Special effort is put on SRAM-based PUFs and delay-based PUFs, particularly the RO-PUF due to their simplicity and overall good performance, as proven experimentally in [42, 43] where the authors show that SRAM PUFs and both delay-based PUFs show good PUF behaviour, with high reliability (less than 10% noise at corner cases and after ageing) and high uniqueness (very close to 50%).

2.1 PUF Taxonomy

Following the classification presented in [44] and the additions of novel technologies as highlighted in [45], PUFs can be classified according to the origin of the randomness source, the evaluation process and the application:

- According to the randomness source as follows:
  - Implicit: They are inherent in the physical characteristics of the device or component itself. They exploit inherent variations in manufacturing processes or physical properties that are not explicitly designed as PUFs but can be used for identification or authentication purposes.
  - Explicit: Explicitly designed and implemented for specific purposes such as authentication or key generation. They are intentionally structured and characterized to provide secure and reliable responses. Randomness is applied externally through additional steps.
- According to the evaluation process:
  - Intrinsic: They derive their uniqueness and variability from inherent physical properties that are difficult to clone or replicate. The variability is primarily due to uncontrollable factors in the fabrication process.
  - Extrinsic: Utilize external stimuli or components to create variability or responses. They may rely on environmental conditions or additional components to generate their unique identifiers.
- Finally, according to the application:
  - Weak: Low number of challenges. Industrial grade PUFs tend to have 128 to 256 CRPs.
  - Strong: Extremely large number of challenges that cannot be used exhaustively.

A summary of the main distinctions between weak and strong PUFs can be found in [46].

An overview of the most common PUF designs as depicted by [47, 44] is portrayed in 3.

PUF Technology	Variations	Classification
Arbiter	CMOS Arbiter PUF [48]	Explicit, Intrinsic, Strong
Ring Oscillator	TERO PUF [40]	Implicit, Intrinsic, Weak
	VERO PUF [49]	-
	Loop PUF [50]	-
	Group-based RO [51]	-
SRAM PUF	Originally proposed in [52]	Implicit, Intrinsic, Weak
DRAM	Originally proposed in [53]	Implicit, Intrinsic, Weak
FPGA Intrinsic	LUT PUF [54]	Explicit, Intrinsic, Weak
	CLB PUF [55]	-
	DD-PUF [56]	-
	Routability PUF [57]	-
Butterfly	Variation of the SRAM PUF tailored to FPGAs	Explicit, Intrinsic, Weak
Memristive	[37], [58], [59], [60]	Explicit, Intrinsic, Weak
	mrPUF [61]	Implicit, Intrinsic, Weak
Magnetic	Magnetic Tunnel Junction [62]	Implicit, Intrinsic, Weak
Carbon Nanotube	Originally proposed in [63]	Explicit, Intrinsic, Weak
Optoelectric	Photonic PUF [64]	Explicit, Extrinsic, Weak

Table 3: PUF Classification

Furthermore, in the recent years Higher Order Alphabet (HoA) PUFs [65] have been proposed, where the output is not limited to bit values but to a series of symbols. However, this still remains a little explored field due to the added complexity of their security assessment [66].

2.2 Ring Oscillator PUF

A Ring Oscillator (RO) consists of an odd number of inverters connected in a ring whose output oscillates due to the delay between different inverters To assure that a RO only oscillates when necessary, an enable signal should be used. This can be done by using a

NAND gate and an even number of inverters, as shown in Figure 1 or an AND gate and an odd number of inverters.

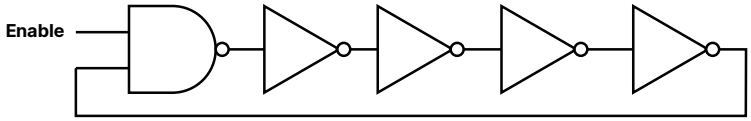


Figure 1: Ring Oscillator Schematic with NAND Gate

A Ring Oscillator PUF (RO-PUF) leverages the unique characteristics of ring oscillators. Due to manufacturing variability the ROs oscillate at slightly different frequencies. When a challenge is applied to an RO-PUF, the pair of ROs given by the challenge oscillates, and the resulting response is computed based on the frequency difference on the pair as depicted in Figure 2. Consequently, all CRPs of an RO-PUF is extracted by measuring the frequency difference of all available pairs of ROs.

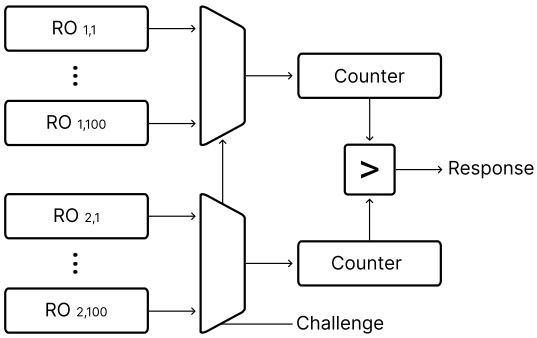


Figure 2: Design of Ring Oscillator PUF under study

RO-PUFs are advantageous due to their simplicity and ease of integration into existing digital circuits. They are also known for their robustness against environmental changes, making them a reliable choice for many applications.

However, RO-PUFs are very sensitive to supply voltage variatations, which is one of the main reasons of unreliability. Furthermore, there is an relationship between the number of stages and the quality of the bits generated, as demonstrated by [67], so a proper assesment of each RO-PUF desgins is necessary.

2.3 SRAM PUF

SRAM-based PUFs exploit the power-up states of SRAM cells to generate unique identifiers. While there are many designs of SRAM cells, and topologies, the most common one remains the 6 transistors (i.e. 6T) shown in Figure 3, as it is the fastest.

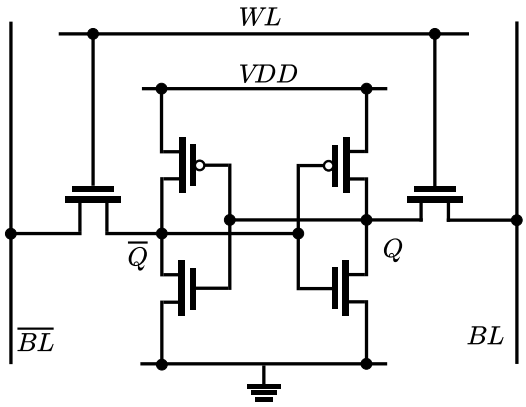


Figure 3: Schematic of a 6T SRAM Cell

While each cell is designed to be symmetric, process variability will result in inverters of different strength, making the cell biased towards 0 or 1 so that when an SRAM cell is powered on, it randomly settles into a 0 or 1 state. By reading the initial states of a set of SRAM cells, a unique fingerprint can be generated. In this case, the set of CRPs of the SRAM-PUF is extracted by reading the required number of bits.

SRAM-PUFs are particularly appealing because they do not require any additional circuitry; they use the existing memory cells present in most microcontrollers and processors. This makes them cost-effective and easy to deploy in a wide range of devices.

One of the first to exploit metastability effects is the work in [68]. Following that, many statistical and empirical analysis on the stability of SRAM cells [69, 52, 70, 71] suggested the use of SRAM cells for PUF designs. Among the numerous silicon-PUFs proposed in the literature, the SRAM-PUF [72, 35, 52, 39, 73] is one of the most popular because ubiquity of memories in any circuit. Due to transistors' manufacturing variations the symmetry of an SRAM cell is broken resulting in a preferred power-up state for the memory cell. Combining multiple cells is possible to create a start-up pattern that can be used as a signature. The Butterfly PUF [74] uses the same principle of the SRAM PUF, but is usually deployed on FPGAs using LUTs.

One of the main limitations of SRAM-based PUFs is the low yield of reliable bits as shown by [75], which most of the time is solved with ECCs and a large area consumption due to the redundancy. Although unwanted, this effect can be exploited to generate random numbers as depicted in [76, 77].

SRAM-PUFs are very sensitive to supply voltage as the cell power-up state is strongly dependent on the power supply ramp rate and direction as suggested by [78] and demonstrated through experiments and simulations [79, 80]. Many power control techniques have been proposed in the literature as seen in [81, 82, 83, 84]. However, this still remains a great

challenge as normally SRAMs are also shared by other applications and a well-designed power system needs to be used across all applications.

Challenges in PUF design

I.3

PUFs designs have gathered significant attention from the research community, particularly in identifying vulnerabilities and in parallel have raised some important questions regarding their quality and security. A comprehensive overview of PUF attacks and limitations is presented in[1][2], with many novel attack techniques and mitigation strategies emerging since then.

Side-channel attacks exploit implementation-specific phenomena to extract information about certain behaviours of interest. For instance, side-channel attacks have yielded promising results in arbiter [87] and SRAM-PUFs [88] through power analysis. A plethora of mitigations techniques [89, 90, 91], have been proposed to combat a range of non-invasive and probing attacks.

The most compelling attacks challenging the unclonability claim of PUFs are invasive techniques, as demonstrated in [92, 93], where a series of invasive optical attacks successfully extracted SRAM bits and facilitated the creation of custom SRAM clones. However, these type of attacks still remain very costly and require expensive machinery with limited availability.

Moreover, external attacks are not the sole concern for PUFs. Environmental factors significantly impact the reliability of responses. Techniques to mitigate the effects of environmental factors are proposed in [94]. Another critical limitation is the reliability of PUFs, specifically their ability to consistently produce the same secrets.

Numerous studies have attempted to quantify the effect of ageing on PUF reliability, such as [95, 96]. These works observed that most of the PUF's ageing instability occurs early in its lifetime. Additionally, a high correlation has been observed between instability caused by ageing and instability caused by temperature.

The work in [97] shows that ageing has a minimal impact on delay chains, as each element ages independently. However, the memory point, such as the latch of the arbiter, is much more sensitive to ageing due to the asymmetry of its dual structure. Thus, the ageing of an element differs from that of its dual element, and the difference continuously increases. This highlights the advantage of using simple delay-PUFs, such as the Loop-PUF, to avoid the imbalance of the arbiter or SRAM memory points. Promising modifications to certain PUF designs have been proposed to combat ageing and reduce the effect of external working conditions [98].

[1] D. Mukhopadhyay and R. S. Chakraborty, *Hardware security: design, threats, and safeguards*. CRC Press, 2014.

[2] U. Rührmair *et al.*, 'Special session: How secure are pufs really? on the reach and limits of recent puf attacks,' in *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2014, pp. 1–4. DOI: 10.7873/DATE.2014.359.

From an analytical and software perspective, reliability issues are typically addressed through Error Correcting Codes shortplural (ECCs), as shown in [99, 100]. Additional software-based techniques have been proposed to complement ECCs [101]. However, error-correcting mechanisms could leak critical information about the PUF, as illustrated by new threats regarding helper data leakage and manipulation [102, 103]. One important example of this is highlighted in [104], where the authors show that an attacker can observe system failure rates of an RO PUF to retrieve the public key, or at least obtain some information about it. To mitigate these issues, the authors we encourage the use of fuzzy extractors, the well-established reference solution.

Implementation-specific limitations also exist. For example, [105] demonstrates that when a PUF requires routing constraints, as with arbiter PUFs, its implementation is more effective on Application-Specific Integrated Circuit (ASIC) than on FPGA due to manual routing. Additionally, Complementary Metal Oxide Semiconductor (CMOS) variation extraction is better on ASIC than FPGA, enhancing PUF uniqueness. Experimental studies like [106] and [107] show that variability distribution in FPGAs is not uniform, necessitating extreme care in the placement of ROs in the FPGA and their comparison method.

Modeling attacks present a significant threat to PUFs, particularly to strong ones. Advanced Machine Learning (ML) algorithms and sophisticated statistical techniques have demonstrated their efficiency in modeling the behavior of strong PUFs, enabling attackers to predict previously unseen responses. Additionally, both invasive and non-invasive methods have proven effective in replicating weak PUFs. A comprehensive description of these modeling and cloning attacks is provided later in Chapter I.1.

In conclusion, the challenges that hinder PUF large-scale deployment are:

- Side-channel and invasive attacks pose serious threats to PUFs, with mitigations and new attack techniques continually emerging.
- Environmental factors and ageing impact PUF reliability, with studies highlighting early instability and high correlation between ageing and temperature effects.
- Reliability issues are addressed through ECCs and software-based techniques, though these can introduce new vulnerabilities. Fuzzy extractors seems to solve these issues without introducing vulnerabilities.
- Developing protocols for PUFs is challenging, requiring new hardware properties and certification mechanisms.
- The lack of standardized certification slows down broader adoption of PUFs in security-sensitive applications, highlighting the need for tailored certification processes.
- ML and statistical techniques pose a significant threat to strong PUFs and invasive techniques have been proven effective in creating clones of weak PUFs.

The scope of this manuscript is narrow, and not all issues are addressed. In this thesis, we concentrate on the Reliability of differential PUFs, particularly Ring Oscillator PUFs (RO-PUFs), and its relationship with Entropy. We also conduct a detailed study of the canonical metrics for PUFs, their interconnections, the proposal of new metrics to examine

rarely studied effects such as spatial correlation, and the development of methodologies to estimate the behavior of a PUF system from a small sample set. Additionally, we explore the use of mathematical analysis to create “digital twins” or software clones of certain PUF designs, which reduces the cost and time required for PUF testing.

II

Simulation Workflow

Simulation Environment

II.1

Design complexity, ultra-low-power requirements, reliability, robustness, and security are becoming increasingly important concerns in electronic system design. Additionally, the aggressive scaling of CMOS technology and the introduction of beyond-CMOS devices (such as CNT-FET, Memristive, and Spintronic) impose additional challenges, necessitating thorough testing to account for varying technological characteristics. Studying the behaviour of electrical circuits under different conditions and parameters is imperative. Multiple simulations are performed under the appropriate conditions to achieve this. In today's circuits with novel technologies, the number of parameters and their interdependencies can be prohibitively large for exhaustive study, making statistical techniques like Monte Carlo[1] and parametric simulations the classical approach to obtain statistically relevant data for proper circuit assessment. SPICE[2] is the lingua franca used to describe electronic circuits and is at the heart of every electric simulator.

The circuits under study were created in Cadence Virtuoso and simulated using Cadence Spectre, with all designs using the CMOS 65 nm ST Microelectronics technology. Verilog-A modules were developed to record information about parameters of interest at specified intervals. The Verilog-A source code of these modules is provided in the Appendix B.3. Spectre stores the rest of the parameters and simulation results by default into a PSF database (a proprietary format of Cadence). Additional post-processing was necessary to export the selected signals and parameters to a format suitable for the analysis environment. With the inclusion of various Verilog-A modules, the signals and parameters were directly stored in CSV files, which were better suited for analysis.

Ocean, the proprietary scripting language developed by Cadence, was used to perform simulations and to perform computations (i.e. oscillation frequency). During the initial iteration cycles, simulations were launched using a combination of shell batch jobs and Python scripts. While effective for initial simulations, the need for in-depth analysis required the development of more sophisticated tools offering greater flexibility. An overview of the initial environment used is shown in Figure 4.

Nonetheless, several limitations of the Electronic Design Automation (EDA) and simulator software severely restricted the development of the studies.

First, the utilized Probability Density Function (PDF). The absence of these models restricted access to specific EDA tools, including Monte Carlo simulations. To overcome this limitation, it was necessary to either develop the statistical models from scratch or directly inject the

[1] N. Metropolis and S. Ulam, 'The monte carlo method,' *Journal of the American statistical association*, vol. 44, no. 247, pp. 335–341, 1949.

[2] L. W. Nagel and D. Pederson, 'SPICE (simulation program with integrated circuit emphasis),' EECS Department, University of California, Berkeley, Tech. Rep. UCB/ERL M382, Apr. 1973. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/1973/22871.html.

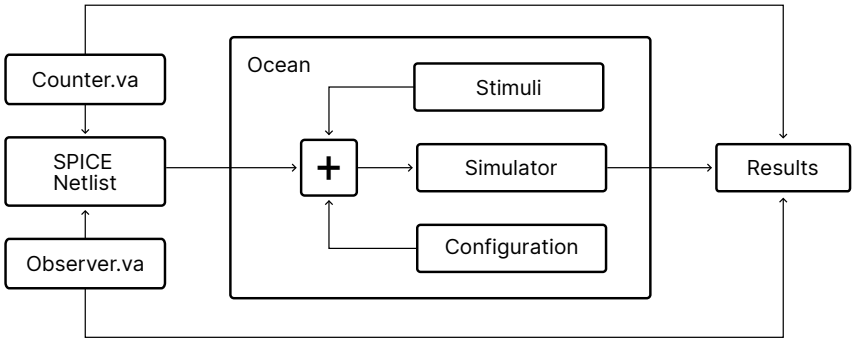


Figure 4: Diagram of the Simulation Environment

parameters into the netlists. This process was highly time-consuming, particularly when simulating various circuits with differing parameters.

Second, the EDA software exhibited a significant constraint: it had a fixed upper limit of 1024 parameter variations per simulation. While this might be adequate for very small circuits or a low number of parameters, it proved insufficient for our study of ROs, which easily exceeded this limit. The RO under simulation, illustrated in Figure 2, consists of a NAND gate and an even number of inverters. The NAND gate comprises four transistors, and each inverter consists of two transistors. For each transistor, the width, length, and threshold voltage were varied. Considering a minimum of three values per parameter (minimum, maximum, and nominal), the total number of variations is calculated as $(4 \cdot 3 \cdot V) + (N \cdot 2 \cdot 3 \cdot V)$, where N is the number of stages and V is the number of values. For example, with $N = 64$ and \$V = 3 \$, the total number of values is 1188, which exceeds the upper limit. In practice, we would need hundreds of values per parameter and many more stages. Therefore, performing parametric simulations directly using the EDA software was highly constrained, necessitating the use of batch jobs and Python scripts as workarounds.

Another significant issue was the lack of observability. Although Virtuoso can save all parameters during the transient simulation onto a PSF database, manually exporting this data can delay the analysis workflow. Additionally, several critical requirements imposed serious constraints on the workflow.

Exporting data to custom formats further complicated the workflow. Aggregating results after complex parametric simulations is tedious and time-consuming, requiring a significant portion of project time just for data preprocessing and preparation.

The scarcity of open-source tools for PUF analysis, particularly within the realm of EDA, necessitated the development of new open-source solutions. Notably, the availability of tools between industry and academia, as well as between digital and AMS designs, varies significantly. Observing the landscape of analog tools for security revealed an opportunity to advocate for the utilization of open-source tools by developing and introducing new ones.

Finally, the absence of standardized software and algorithms for PUF analysis posed a significant challenge as well. Despite the existence of canonical algorithms recognized by the scientific community, there is a notable lack of standardized and publicly documented software for PUF analysis. While there are established test suites for RNGs, such as NIST and Dieharder, most PUF analysis algorithms lack similar standardization. To address this gap, the open-source PUFR (available online at https://servinagrero.github.io/pufr/) library was developed for the R programming language. This library was consistently used for the statistical evaluation of the PUF under study. Future plans include expanding this suite of functions and developing a Python version.

These limitations and constraints prompted the creation of numerous open-source frameworks and data analysis libraries. These tools were designed to be easily adapted to the existing environment and are expected to significantly benefit the analog design community by addressing the identified challenges.

Regarding the development of new tools, it is crucial to address the exploration-exploitation dilemma, a concept relevant across various scientific domains. This dilemma encapsulates the balance between two contrasting strategies: exploitation and exploration. Exploitation involves selecting the best option based on current knowledge of the system, which may be incomplete or potentially misleading. On the other hand, exploration entails experimenting with new options that might lead to superior outcomes in the future, though this comes at the expense of immediate exploitation opportunities. Essentially, there is a trade-off between extensibility (addressing a wide range of problems) and specificity (effectively solving a particular problem).

The following sections present in detail the various software tools developed to tackle the limitations of the study presented here.

Monaco

II.2

To explore the behaviour of an electrical circuit under different designs and conditions, multiple iterations and simulations must be performed under the desired configuration. The interdependencies of large and complex circuits can quickly become a significant challenge due to the extensive number of choices at play.

Design Space Exploration (DSE) examines the various possibilities and design options within the allowed design space, considering the constraints and requirements to fulfil specified performance goals. DSE typically involves the use of tools and high-level abstract models to automate and streamline the exploration process, as the design space is too vast to be explored manually. The industry is keen to accelerate this process and reduce the time between iteration cycles. Computer Aided Design (CAD) and EDA have drastically improved over the last decades, thanks to new methodologies, tools (e.g., Cadence, Synopsys, Xyce), and the recent addition of artificial intelligence techniques such as genetic algorithms and machine learning.

However, algorithms implemented in industrial CAD tools do not offer designers full control and observability of desired parameters. Additionally, the range and distribution of parameters are defined in technological libraries provided by silicon foundries. These data are not easily accessible for emerging technologies and are not yet embedded in industrial libraries.

A unified framework is proposed for both statistical and parametric simulations performed independently or concomitantly. This framework allows designers to fully control the range of parameters and observe circuit behaviour comprehensively. It also enables the correlation of circuit behaviour with the parameters used during simulation. The framework implements the following steps:

- Define the set of parameters relevant to the study, targeting process variability, operating/environmental conditions, stochastic behaviours, ageing, and perturbations.
- Identify the relevant signals that fully characterize circuit behaviour.
- Generate the value of each parameter using a user-defined function.
- For each combination of parameters, generate and simulate a netlist. The relevant signals provide the required information about circuit behaviour.

2.1 State of the art

The majority of available EDA software for electrical design and simulation provide a graphical user interface to generate circuits based on a drag-and-drop mechanism which is very user-friendly. The graphical user interface can be also used to customize different aspects of the simulation to provide the necessary parameters to the simulator.

However, not every available tool provides a mechanism to easily save this configuration to be used later, or on a different computer. Therefore, in order to share a project between different teams, the exact same configuration of the project and sometimes even the operating system have to be precisely copied, which is an unreasonable task to do. Moreover, this problem grows larger with the complexity of the circuit.

CAD tools automatically perform the conversion between the graphical schematic and the textual netlist. Moreover, there are some tools available online that allow parsing netlists or converting results from one file format to another, but these tools are limited since parsing netlists is contextual and simulator-dependent. The same is applied for the file formats containing simulation results as most of the time a special tool is needed to view the results, which makes aggregating results challenging. An example of an automated CAD tool is the AWR Design Environment (AWRDE), provided by Cadence, working in a custom language called SAX. This tool works via a simple interface where the user can control any aspect of the environment through code. Users have also access to powerful tools that allow them to modify part of the circuit and generate custom reports programmatically, and even perform analysis in Python. However, this tool requires an expensive licence that not everyone is able to afford and requires some time to learn to properly use its functionalities.

Additional tools like PySpice[1] and Skidl (https://devbisme.github.io/skidl/) allow the generation of netlists and the launch of simulations directly with Python. This integration consolidates both stimuli and configuration within a unified logic block, thereby minimizing the time required for iteration cycles. However, it is important to note that this solution is confined to the NGSpice and Xyce simulators, potentially influencing the overall workflow. This however, has been one of the main inspirations to build the software presented in the next chapter. As of recent works, the authors in[2] also resort to the methodology proposed here.

2.2 Proposed methodology

The proposed workflow using this tool is the following:

1. Generate the design to study using the EDA tool of choice
2. Obtain the necessary design files. Most likely just the SPICE netlist will suffice
3. Modify the files to add the placeholders where the values will be injected
4. Generate the inputs and stimuli that we want to inject
5. Provide the configuration to Monaco, that is, the number of iterations, the stimuli, the templates and the commands to execute
6. Monaco will take the templates, inject the values and execute the commands in order for the specified number of iterations

[1] F. Salvaire, *Pyspice*. [Online]. Available: https://pyspice.fabrice-salvaire.fr.

[2] A. Xynos and V. Tenentes, 'Metaspice: Metaprogramming spice framework for the design space exploration of puf circuits,' in *2023 12th International Conference on Modern Circuits and Systems Technologies (MOCAST)*, 2023, pp. 1–4. DOI: 10.1109/MOCAST57943.2023.10176643.

An advantage of using template engines is that control logic can be embedded directly inside the templates, which in turn means that certain parts of a template can be rendered or discarded depending on the values of some variables.

2.3 Description of the framework

The framework is divided into two parts. The first part is a spice netlist and simulation configuration builder written in Python. The second one is a Verilog-A module to write the state of an electrical component to a CSV file.

The process of parametric simulations boils down to creating a copy of the analog design and "injecting" the desired parameter values into their corresponding place. This can be done directly from ADE XL in Cadence, where we have a graphical interface to select the parameters in the circuit we want to change.

And since it's a text format, it's easy to generate them by hand, although it's easier to generate circuits with graphical tools. The first iteration used the Jinja Python library (https://jinja.palletsprojects.com/en/2.10.x/) to inject the parameter into text files. The project was ported to Rust, and it now makes use of the Handlebars template engine (https://handlebarsjs.com/).

2.4 State observer

The state observer is designed for Spice- and Verilog-A-based simulators. It is a sub-circuit that is placed in the schematic and is connected to the components in need of measurements.

For transients simulations, the state observer will write the state measured every time step to a CSV file, so that it can be easily loaded into any conventional software to analyse and extract information. For this reason, the simulation parameter `STEP_SIZE` found in the simulation environment is particularly important: it allows running the state analyser with the desired granularity. Moreover, as shown in the examples section, the state observer can be modified to save the wanted data only in certain cases, to allow a lighter and more application-specific log.

The idea of being able to externally save results in a user-defined file comes also useful in case we want to modify the parameters for the next simulations according to the results just obtained. As an example, we could increase or decrease the step size of parameters depending on how close we are to the expected working behaviour.

As previously illustrated, the capability to export desired results directly to a CSV file offers two distinct advantages. Firstly, it eliminates the need for supplementary software to convert data into a suitable format for analysis. Secondly, it affords complete control over

the data and its formatting, thereby reducing friction and facilitating greater extensibility within our analysis workflow.

The Verilog-A module is provided in the Appendix B.4.

2.5 Netlist and simulation configuration tool

Monaco leverages the textual nature of netlists to generate the necessary stimuli for simulations. A set of parameters can be generated a priori, injected into the templates, and fed to the simulator. To achieve this, the user needs to define the different stimuli files used as templates and specify the functions to generate the values for each iteration. The input files are treated as templates and processed by a template engine. An advantage of using a template engine is that logic can be embedded directly into the templates, allowing certain parts of a template to be rendered or discarded based on variable values. A project can be configured either in a TOML or a YAML file, as shown in 2.1. The actual configuration file used for the simulations performed during this thesis is shown in the Appendix B.5.

```yaml
jobs:
  - name: oxram
    ignore_errors: false
    iters:  "./props.json"
    log: "./log/run_{{iter.idx}}.log"
    templates:
      - "template.scs:input.scs"
      - "runTemplate:runSimulation"
    steps:
      - sh ./runSimulation
```

Listing 2.1: Configuration example for Monaco

The source code presented in 2.2 shows a small section of a SPICE netlist featuring a Ring Oscillator. This part generates a chain of inverters where the number of stages is determined by the user variable `nstages`. In this example, variability is induced by altering the length and the V_{th} of each PMOS and NMOS. Thanks to the Jinja templates, the number of inverters for each Ring Oscillator can be embedded in the logic and configured by the user. However, this approach can quickly become convoluted if the requirements change throughout the verification process.

```
V0 (vdd 0) vsource dc=${vdd} type=dc

{% for s in range(1, props.nstages) -%}
I{{loop.index}} (0 net{{s}} net{{s-1}} vdd) INV \
    vthp={{ params['vthp'][loop.index0] }} \
    vthn={{ params['vthn'][loop.index0] }} \
    lp={{ params['lp'][loop.index0] }} \
    ln={{ params['ln'][loop.index0] }}
{% endfor -%}
```

Listing 2.2: Jinja template to generate inverters. The Inverter subcircuit definition has been omitted.

2.6 Conclusions

The framework shown in this thesis is able to generate complex parametric electrical simulations in a reliable and repeatable manner. This framework is open-source under the MIT licence and works with any available commercial simulator as it doesn't require major modifications to the files used. This project was published in [1] and can be found online at https://servinagrero.github.io/monaco.

NIMPHEL

II.3

The proposed framework aims to provide utilities that enable quick design space exploration and parameterization of electronic designs. It offers a high-level abstract interface to create modular and reusable components, which can be seamlessly parameterized to give users a comprehensive overview of the design under various constraints and environments.

Using a programming language like Python as an abstraction layer for circuit generation eliminates the limitations of a drag-and-drop graphical user interface, allowing the expressiveness of Python to quickly generate complex structures and leverage numerous available scientific libraries. Similar to how CHISEL uses Scala to generate Verilog, this approach benefits from the established language ecosystem, with limitations confined to those of the language itself. While creating a custom file format is another solution, it involves higher friction as users would need to adapt or develop new simulators to accommodate the new format.

Graphical interfaces are prone to changes over time, whereas programming languages remain stable, reducing the need for users to learn different software versions. Design changes are reflected in the source code, simplifying the versioning process. Additionally, using a well-debugged and formalized tool eliminates the need to manually check finished netlists, a necessary step when netlists are created by hand and require validation and correction for every modification.

Most commercially available software provides interfaces for parametric analysis of a design. However, generating different versions of a circuit, such as a flash ADC with varying bit numbers, typically requires manual creation, limiting exploration due to time and complexity constraints. This tool allows embedding parametric characteristics directly into components, enabling modular and programmable generation of multiple designs.

Moreover, most automatic circuit generation tools are either closed-source or application-specific, such as SRAM generators. Python tools like PySpice and Skidl are primarily focused on PCB design, which falls outside the scope of this framework.

The framework has two primary objectives: (i) to provide tools for creating electrical components whose characteristics can be defined through dynamic models or logical rules, and (ii) to offer powerful manipulation primitives for quickly creating complex component arrangements. Leverageing Python's utilities and flexibility, the framework facilitates the generation of modular and reusable components. Designs created in Python can be converted to any text format specified by the user, with a particular focus on SPICE netlists. Although the framework emphasizes the rapid generation of complex designs, users can write extensions for automatic simulation of the generated designs or other tasks such as Electrical Rule Checking (ERC).

3.1 State of the art

Despite some incremental advancements in EDA tools, there has been a lack of significant innovation or the introduction of new methodologies within the academic community. In contrast, the field of digital electronics has seen the development of a vast array of tools and methodologies.

Numerous tools are available for design space exploration. Frameworks like Chisel[1] and PyMTL3[2] offer high levels of abstraction, enabling the compilation of high-level languages such as Scala and Python into fully functional Verilog code for hardware description. These tools empower circuit designers with the expressiveness and power of programming languages, facilitating the rapid creation of reusable circuits. However, these tools are primarily targeted at Register Transfer Level (RTL) design and are not well-suited for analog and mixed-signal designs.

PySpice [110] is a utility for generating SPICE netlists and launching simulations, embedding the design and simulator configuration within the same language to streamline the design iteration process. However, its simulation capabilities are limited to NGspice and Xyce, and netlists can only be exported for PCB designs. Skidl (https://github.com/devbisme/skidl) is built on top of PySpice, aiming to simplify the process of connecting different components. The work in [114] also presents a similar methodology by providing a Python interface to SPICE-based simulations. SPICE netlists are a universal format used by electronic simulators, though each simulator has slight variations. The proposed framework aims to support designs for any available simulator by providing tools to export the designs accordingly.

Projects such as Magic[3] provide automatic layout generation mechanisms. Magic is an interactive software for creating and modifying VLSI circuit layouts, with a key feature being the ability to scale layouts for different technology nodes. While user-friendly, this ease of use comes with a 5 to 10% increase in area usage. Tools like LibreCell (https://codeberg.org/librecell) attempt to reduce this trade-off by limiting user options. Lower-level tools such as GDSTK (https://github.com/heitzmann/gdstk) and GDSFactory (https://github.com/gdsfactory/gdsfactory) facilitate the creation and manipulation of GDSII and OASIS files, the standard formats for specifying circuit layouts in foundries. These tools can serve as the foundation for more comprehensive software capable of generating layouts based on circuit definitions.

Researchers have also focused on intelligent methodologies for automatic layout generation for both PCBs [116, 117, 118] and ASICs [119]. Genetic and evolutionary algorithms have proven effective for these tasks [120].

[1] J. Bachrach *et al.*, 'Chisel: Constructing hardware in a scala embedded language,' in *Proceedings of the 49th Annual Design Automation Conference*, 2012, pp. 1216–1225.

[2] S. Jiang *et al.*, 'Pymtl3: A python framework for open-source hardware modeling, generation, simulation, and verification,' *IEEE Micro*, vol. 40, no. 4, pp. 58–66, 2020.

[3] J. K. Ousterhout *et al.*, 'The magic vlsi layout system,' *IEEE Design & Test of Computers*, vol. 2, no. 1, pp. 19–30, 1985.

AIDA [121] addresses analog IC sizing and layout, offering utilities for parametric analysis, allowing users to generate and swap circuit parameters before each simulation cycle. However, it requires the user to generate the design beforehand, limiting its utility for design exploration.

Complete projects like OpenRAM[4] provide a Python framework for creating layouts, netlists, timing and power models, and placement and routing models for using SRAMs in ASIC design. OpenRAM offers an easy interface for configuring SRAM characteristics, but it is limited to SRAMs and a few technology nodes (currently NCSU FreePDK 45 nm, MOSIS 0.35 µm, and Skywater 130 nm).

Other literature, such as [123, 124, 125], demonstrates optimization and netlist generation cycles for specific designs. However, these tools are ad-hoc solutions and are not extensible to other designs.

Moreover, electrical components can be imported from various file formats, and designs can be exported to different SPICE formats suitable for any available simulators.

3.2 Proposed framework

Instances of electronic components, as defined in SPICE, can be effectively represented using Python dictionaries. The framework introduces *Component* objects that users configure and subsequently employ to generate multiple instances, akin to the drag-and-drop mechanism of existing EDA software. Beyond basic properties like component name, connected nets, and parameters, users can embed metadata to provide additional information that can be shared across different tools. These components act as templates to generate modular circuits, treating electrical components as black boxes with inputs, outputs, and parameters. Verilog-A described components can also be utilized seamlessly, provided the simulator supports them. Similar to other software, NIMPHEL offers *Subcircuit* and *Circuit* objects to create modular and reusable designs.

The framework includes parsers for automatic serialization and deserialization of electronic circuits. Currently, the framework supports netlists in Spectre and SPICE formats, as well as Verilog-A modules. Since parsing and translation are independent processes, users can read data from one SPICE format and output their new design in a different format, facilitating quick prototyping.

This capability allows for the progressive adoption of the framework by users and easy transitions between different architectures. The same complex circuit can be generated using various basic cells by providing netlists that define the same subcircuit in different architectures.

[4] M. R. Guthaus *et al.*, 'Openram: An open-source memory compiler,' in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, IEEE, 2016, pp. 1–6.

3.3 Comprehensive Example

The example showcased in 3.1 demonstrates most of NIMPHEL's functionalities by building a resistive array.

As with any Python program, dependencies are first imported. The array topology is then defined, and the different nets are created. The components to be used are created next, including the voltage source, resistors, and ADC. Due to the parametric nature of the resistive array, resistors can be instantiated directly within a single for loop, configured with the variables defined at the beginning.

```
#!/usr/bin/env python3
from nimphel.core import *
from nimphel.readers import *
from nimphel.writers import *
import numpy as np
from itertools import product

# Constants
rows = 20
cols = 10

nets_in = [f"IN_{i:03d}" for i in range(rows)]
nets_col = [f"COL_{i:03d}" for i in range(cols)]
files = [f'"./INPUT_{i:03d}.txt"' for i in range(rows)]
corner_file = "/path/to/corners.scs"

# Components
Vsource = Component("vsource", ["VDD", "GND"], {"type": "pwl"}, cap="V")
Res = Component("resistor", ["P", "N"])

circuit = Circuit()

# netlistHeader
circuit += Directive("simulator", lang="spectre")
circuit += Directive("global 0 gnd!")
circuit += Directive(f'include "{corner_file}"')

circuit.add(
    [
        Vsource.new(dict(VDD=i, GND=0), params={"file": f})
        for i, f in zip(nets_in, files)
    ]
)

for i, o in product(nets_in, nets_col):
    r = np.random.uniform(0, 1e3)
    inst = Res.new(dict(P=i, N=o), params={"r": r})
    circuit.add(inst)

# Sense amplifiers
ADC = Component("ADC", ["IN", "OUT"], {}, cap="I")

for i, c in enumerate(nets_col):
    inst = ADC.new(IN=c, OUT=f"ADC_{i:03d}")
    circuit.add(inst)

writer = SpectreWriter()
```

```
netlist = writer.dump(circuit)
print(netlist)
```

Listing 3.1: NIMPHEL example code to generate a 2D array

3.4 Conclusions

The framework proposed here provides tools focused on fast design space exploration and modular and re-usable electronic designs. Electronic circuits are modelized through the use of Python objects that allow for easy manipulation and quick iteration cycles. The examples provided above show how the framework excels at generating modular architectures and can adapt to multiple technologies and devices. Moreover, the electrical components can be imported from a plethora of file formats and the designs can be exported to various SPICE formats suitable for any available simulator.

While NIMPHEL excels at parametric circuits, like chains or arrays in terms of number of lines of python code to number of netlist lines, it falls short for non-parametric circuits or "special" designs this tool does not offer any advantage. However, it does offer the possibility of changing parameters and performing optimization algorithms although in that case most simulators already offer this functionality.

This project is also open source under the MIT licence, was published in [2] and can be found online at https://servinagrero.github.io/nimphel.

Conclusions

II.4

This chapter has provided an in-depth examination of the simulation requirements along with the workflow used throughout the thesis. The initial needs and challenges encountered are outlined, with key problems identified as limitations within the EDA environment used and the various strategies to address these issues are also discussed. The primary limitations inherent in the EDA software that hindered the studies presented in this document were:

- The cumbersome nature of creating multiple parametric simulations across different circuits and requirements.
- The physical limitation of certain EDA software on the number of parameters that can be modified during parametric simulations.

To overcome the limitations two open-source tools were developed and utilized. The first tool, Monaco, addresses the constraints of parametric simulations in major EDA platforms. Monaco functions similarly to templating software, enabling the injection of user-generated values into files and executing specified commands with those files, thereby enhancing the flexibility and efficiency of the simulation process that is tailored directly by the user. This framework is open-access and open-source and works with any available commercial simulator as the parametric injections are performed directly on the files that are fed to the simulator.

While this tool effectively facilitated a large series of parametric simulations, it became cumbersome when different circuits were to be simulated. As previously mentioned, each user needs to create templates for the injection of parameters. Although this process can be automated using certain tools and advanced template usage allows for specific design modifications, it is not suitable for exploring various designs, as it was the case with the different configurations of Ring Oscillators that were studied.

To address this, the concepts of this framework were extended and combined with ideas of other tools in the literature of digital design that generate analog netlists in a parametric way. The second tool, named NIMPHEL, is a Python framework designed for the programmatic generation of SPICE netlists. Comparable to the CHISEL library in Scala, but tailored for Analog and Mixed Signals (AMSs) designs, NIMPHEL simplifies the creation and manipulation of SPICE netlists, facilitating more sophisticated and automated simulation workflows. This approach leverages the full capabilities of a programming language like Python, allowing for the programmatic debugging and formalization of generated circuits. In contrast, manually created netlists require checking and correction for each modification.

NIMPHEL enables the creation of subcircuits or certain instances from another tool (e.g. Cadence), parsing pre-existing circuits from netlists, in-place modification (e.g. process variability simulations) or extension by creating circuits that depend on user defined configuration

(e.g., macro compilers for SRAM, MACs, chains, etc). It is designed not to replace commercially available CAD tools but to enhance designer productivity by simplifying the process of working with modular circuits.

Future work will focus on the automatic generation of layouts across various formats and technology nodes. Integrating tools such as ASG[1] and N2S[2] with this framework could facilitate the creation of schematics from the generated netlist. Additionally, employing well-known frameworks like Magic [115] to automatically generate circuit layouts based on the netlist and a set of predefined rules could significantly accelerate the design process, allowing for easy generation and optimization of circuits. Another promising approach to enhance and speed up the design phase is the application of machine learning techniques [127], particularly genetic algorithms, which have demonstrated considerable effectiveness in topology optimization [119, 118, 117, 120].

[1] Y.-S. Jehng *et al.*, 'ASG: Automatic schematic generator,' *Integration*, vol. 11, no. 1, pp. 11–27, 1991.

[2] B. Naveen and K. Raghunathan, 'An automatic netlist-to-schematic generator,' *IEEE Design & Test of Computers*, vol. 10, no. 1, pp. 36–41, 1993.

III

Circuits under test

III.1 Simulated and Manufactured PUFs under test

Access to both experimental and simulated data is crucial for comprehensive evaluation of PUFs and electronic circuits in general. Experimental data serves as a means to validate the accuracy and reliability of simulation models and methodologies. While simulations provide insights under ideal conditions, real-world data enables verification against actual performance, facilitating iterative refinement of models. Additionally, empirical data captures nuances like environmental variations, manufacturing imperfections, and transient phenomena, which simulations may overlook. Incorporating experimental data into evaluation methodologies enhances understanding of the combined impact of manufacturing variation, design, and environmental conditions, enabling more effective PUF analysis, particularly in assessing reliability, a challenging metric to evaluate.

The platform presented in this chapter was developed with the goal of facilitating access to a substantial volume of SRAM data, enabling the robust validation of our reliability claims concerning SRAM-based PUFs. Later on during the development of the thesis, access to extensive RO data was grated by Infineon. Despite the fact these ROs were not originally designed for the specific purpose of RO-PUF implementation, the dataset provided considerable flexibility, enabling the adaptation and validation of our analysis to enhance the performance of the resulting PUF. In this chapter an in-depth description of the dataset obtained is presented. Proper analysis will be presented in Chapter III.3.

SRAM-based PUF Experimental Platform

III.2

Testing the reliability of PUF designs is non-trivial, as it requires extensive testing of numerous devices over prolonged periods and under a wide range of environmental conditions. Traditionally, the impact of ageing and operating conditions has been addressed through simulations [128]. For evaluations on physical devices, particular attention has been given to SRAM embedded in microcontrollers. For instance, studies [35, 52, 39] collected 30 samples from 10 MSP430F1232 microcontrollers to analyse PUF behaviour. In another study [129], researchers examined the SRAM quality of 26 STM32F303 and 31 STM32F407 microcontrollers, collecting 37 samples from each. Additionally, [130] published 200 samples of raw SRAM data from 144 Cortex-M4F devices for further SRAM-PUF research. More recently, the effects of ageing were extensively measured using 16 Arduino boards over two years, resulting in approximately 175 million measurements [131], making this one of the most extensive studies found in literature.

However, the limited number of samples and devices (with the exceptions of [130, 131]) and the lack of public availability of samples have hindered further research on performance evaluation, ageing studies and suitable post-processing techniques. These restrictions motivated the development of the platform presented here, aimed at gathering extensive SRAM data and providing a comprehensive dataset to the scientific community.

Additionally, this platform introduces a novel feature: the ability to write custom values to any SRAM region, facilitating the study of Negative Bias Temperature Instability (NBTI) effects. It has been demonstrated that storing a specific value in an SRAM cell (e.g., 0) can reinforce the tendency to power up to the opposite value (e.g., 1) due to ageing mechanisms [132, 133, 134, 135]. This phenomenon, common in PMOS transistors, increases the threshold voltage, leading to decreased drain current and transconductance. Ageing can cause inverters in SRAM cells to behave differently from their initial measurements, producing unexpected responses. To mitigate this, it is proposed to induce ageing to improve the PUF reliability, thus reducing the number of bit flips in successive readings.

This platform offers significant resource savings for users, both in terms of costs (avoiding the need to purchase hundreds of devices) and time (eliminating the need to collect thousands of samples). The availability of raw data enables users to conduct various experiments, such as designing new post-processing techniques or searching for systematic variations, with a sufficient number of samples and devices to ensure statistical significance. Additionally, the extra device metadata provided, opens up numerous possibilities for detecting vulnerabilities and developing new metrics.

Testing procedure

The different stages of the testing procedures are depicted in Figure 5. The procedure starts by turning on all the devices each Monday morning. To ensure the proper assessment of the SRAM-PUF, a total power-off needs to occur between readings as a software level reset is not sufficient. As it will be explained in detail later, this is performed through specific hardware and a relay. Following this, the station waits 1 full minute until all devices are properly powered on.

Since the devices contain embedded sensors, that information is first read in order to store the conditions at which the readout was produced. Once all the sensor information has been obtained, the platform starts reading all memory regions of all devices in order. Each memory region is stored in the database as soon as it's received to ensure the integrity of each readout. With the current amount of devices under test, this procedure takes approximately 24 hours.

After that, the first half of the devices are in order to study the NBTI effects. The reference sample of each device, is used to craft packets containing the complementary values and they are written to the corresponding region. This procedure takes approximately half a day.

Finally, the boards stay powered on waiting until Friday afternoon where they are powered off until the next Monday where the procedure is repeated.

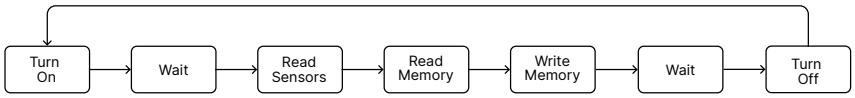


Figure 5: SRAM-PUF testing workflow

The final implementation deployed at TIMA Laboratory in Grenoble, France, is showcased in Figure 86.

Devices under test

The platform evaluation the performance of STM32L152RE and STM32L152RCT6 devices from ST Microelectronics. These prototype devices are equipped with a variety of sensors and General Purpose Input Output (GPIO) ports to enable a variety of communication protocols. In terms of memory, the STM32L152RE has 80 kB of SRAM, while the STM32L152RCT6 has 40 kB of SRAM. Both devices also include FLASH memory, which was not used in this experiment. Nonetheless, the platform and the communication protocol has been designed to be device-agnostic, so different devices can be used in tandem.

To maximize the number of connections, daisy chains with two UART ports for bidirectional communication are employed. The automatic power cycling of the devices is managed

using a YKUSH USB hub (https://www.yepkit.com/products/ykush), enabling independent control of power to three USB ports.

All devices are programmed with identical source code, simplifying the process of adding, removing, and modifying equipment. The physical position of each device is detected using a field in our communication protocol called PIC (Position In Chain). This value starts at 0 to identify the PC and is incremented by 1 for each device downstream. In addition to this position identifier, each ST Microelectronics board has a 96-bit unique identifier (UID) burned into memory (http://blog.gorski.pm/stm32-unique-id). This UID includes the batch number, wafer number, wafer lot, and the X and Y position of the device on the wafer. Together, the PIC and the ST UID provide full control over each device in the chain.

The microprocessors were developed using STM32CubeIDE, with a single version of the code used for all devices, regardless of their position in the chain. This approach makes adding and removing devices highly dynamic, as reprogramming is unnecessary when a device's position changes.

The code for each device is implemented using a simple finite state machine that efficiently utilizes the microcontroller's interrupts to handle bidirectional communication.

The finite state machine operates as follows:

1. Wait for a USART interrupt.
2. If a packet arrives from downstream, send it upstream without reading until it reaches the PC.
3. If a packet arrives from upstream:
    1. Parse the packet.
    2. Increment the PIC by 1.
    3. Depending on the command, send a response upstream or downstream.
    4. Return to step 1.

Each packet is fixed at 512 bytes during compilation but can be modified as needed. To enable automatic packet handling using interrupts, two 512-byte buffers are reserved at fixed memory locations. The first buffer stores packets incoming from upstream, and the second stores packets from downstream. This approach prevents data corruption if packets are sent simultaneously from both directions.

Device manager

On the software level, the platform is composed of 3 distinct modules whose entry point of is named Dispatcher. This component listens to incoming user messages from a message broker and dispatches the appropriate action depending on the message content. A message broker enables applications and systems to communicate with each other and exchange information. It is the primary mechanism that allows a user to send commands to a station for later processing and execution. RabbitMQ (https://www.rabbitmq.com/) is the chosen

message broker. One of the advantages of using RabbitMQ is its access to queues to hold messages, allowing users to send multiple commands at once without waiting for immediate execution.

Once the command has been read from the queue, it is dispatched to the proper device manager, referred to as the Reader. The Reader's objective is to provide an interface for communicating with different devices. It gathers the necessary data from the user message and from external sources, such as a database, and crafts the custom packet to carry out the desired action.

The code was developed to be device-agnostic, enabling a Reader to manage multiple types of devices simultaneously. However, it is recommended that a Reader manage only a specific type of device to avoid unnecessary complexities. Multiple Readers can be assigned to the same Dispatcher, allowing for the execution of commands on a series of devices with a simple group of commands.

Lastly, interfaces were written for logging and database access. Error reporting and long-term storage are imperative, so interfaces for these actions were created. These interfaces allow users to swap the default services for ones that may better fit their environment.

The final architectural design of the platform is represented in Figure 6.

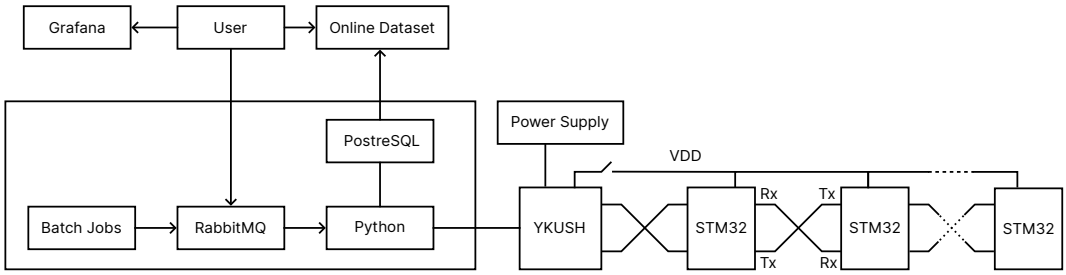


Figure 6: SRAMPlatform architecture diagram

Communication protocol

A bespoke packet-based communication protocol was designed to transfer data efficiently between the PC and the devices. These packets are used to build *atomic* operations that will carry out the designed actions. The packet structure is described in detail in Table 4.

Table 4: Description of a communication packet

Field	Encoding	Description
Command	`uint8_t`	Type of information the packet carries. A full description of all commands is shown on Table 5

Field	Encoding	Description
Options	`uint16_t`	This field has metadata for the command. It may hold the region of memory to read or to write, or which sensors to read
PIC	`uint16_t`	Position in chain of the device in the chain. A value of 0 corresponds to the PC and it gets incremented every time a packet jumps from one device to another downstream
UID	`char[25]`	Unique identifier for the device of 96 bits encoded in a 24 hex characters string
Checksum	`uint32_t`	32 bit checksum to ensure data integrity. Checksum is calculated as the CRC16 of the whole packet
Data	`uint8_t[512]`	Actual body of the packet. It defaults to 512 bytes, but it can be modified before deploying the code

The communication part of the section is implemented in Python through the pyserial module. To send a packet the struct module is used in order to serialize all the fields into binary, as depicted in 2.1, and are sent through the USART port. Once the data arrives into a device, it is then deserialized into a C struct for further processing.

```
packet = Packet() # Generate a packet with default values

# The following are the default values
packet.with_method(Method.Ping)
packet.with_options(0x0)
packet.with_checksum(0)
packet.with_pic(1)
packet.with_uid("DEVICE UID")
packet.with_data([0x0, ..., 0x0])

packet.craft() # Craft the packet to send it

# The packet can now be used by calling `to_bytes`
print(packet.to_bytes())
```

Listing 2.1: SRAMPlatform packet example

The different atomic operations that are currently implemented are depicted in Table 5 and they serve as the basis on which more complex commands are created.

Table 5: Available commands in the platform

Command	Description
ACK	Acknowledgement of an operation. An ACK is sent by a device, for example after a WRITE operation to inform the PC that the command has successfully been executed
PING	This command gives the PC the number of devices that are connected in a chain and their SRAM sizes
READ	Read a region of information of a device. The Options field in the packet contains the offset to read from. The offset is the number of 512 bytes to skip. If the region cannot be read or the checksum does not match, an ERR is transmitted back to the station
WRITE	Write a series of bytes in a memory region of a device. The body in the packet contains the bytes to write. An ERR is sent back to the station in case the checksum does not fit
INVERT	This command employs the WRITE command to deliver the opposite values of the first sample
SENSORS	Extract sensor information from a device. Microcontrollers connected to the platform have temperature and core voltage sensors
LOAD	Load source code to a device that would later be executed
EXEC	Execute code loaded by the LOAD command and store the results into a circular buffer that can later be retrieved
RETR	Retrieve results stored after the code has been executed
ERR	Error during communication. It can be a wrong checksum or a problem during the parsing of a packet

The real implementation of the SENSORS operation is furnished in the Appendix C.1 to see how the interconnection between the different components that make up the platform.

Custom code execution

As seen in the available commands, the platform supports custom code execution via zForth (https://github.com/zevv/zForth), a subset of Forth designed for high portability. The choice of this uncommon language was driven by extreme constraints. Embedding common scripting languages like TCL https://www.tcl-lang.org/, Lua https://www.lua.org/ or LISP https://ecl.common-lisp.dev/ would have left insufficient memory for SRAM analysis. Each device includes an instance of the zForth interpreter, capable of reading code from a buffer, with the buffer size configurable at compile time. Additionally, custom functions to obtain

the current temperature and voltage are provided to the user. For example, the code shown in 2.2 calculates the Fibonacci sequence from 1 to 1024:

```
: fib 1 1 begin .. dup rot rot + dup 1024 > until ; fib
```

Listing 2.2: Fibonacci's sequence in zForth

This code can be executed multiple times without reloading each instance, with results automatically written to a circular buffer for later retrieval. External users can exploit this functionality for their experiments, queuing requests and receiving data directly once processing is complete. This functionality however, has not yet been utilized. The original intention was to allow users to submit their code for evaluation on real hardware. However, the viability of this functionality needs thorough examination due to the security implications of running untrusted code, even in sandboxed environments.

Data Storage

A SQL database is used for reliable long term storage and data retrieval. The interface with the database is designed to be agnostic, so any SQL database should work; for this project we have chosen PostgreSQL. The memory and sensors readouts are stored into two separate tables. Tables 3 and 4 below represent them. Along with sample information, a UTC timestamp is stored with each document to keep track of when the sample was extracted.

Table 6: Common data schema

Field	Description
id	Numeric ID of the sample
board_type	Device model that is connected to the chain. Currently, it's only Nucleo
UID	STM32 96-bit ID formatted as 24 Hex-string.
PIC	*Position In Chain.* Position of the device in the chain, where the first device has a PIC of 1.
created_at	UTC Timestamp when data was gathered in ISO format (YYYY-MM-DD hh:mm:ss)

Table 7: CRPs schema

Field	Description
address	The address of each memory region where the data was read. Each memory region contains 512 bytes. For Nucleo devices, there are 160 regions in total.
data	512 bytes as unsigned integers of the memory region. The values are separated with commas and surrounded with double quotation marks

Table 8: Sensors schema

Field	Description
temperature	Temperature of the device in Celsius
voltage	Internal voltage of the device in Volts

Data monitoring and validation

The data provided by this platform has not been pre-processed to keep raw data of the SRAM. Nevertheless, it is still important to ensure that the data provided does not present faulty bits. To monitor the status of the communication at any given time, each packet contains a header with metadata. To prevent bit flips during data transmission, every packet comprises a CRC16 checksum. Additionally, since every operation the platform performs is atomic, any problems that may occur at any given moment can easily be located and reported. As an example, the following table portrays potential issues that could be detected while performing a WRITE operation on a device.

Table 9: Potential problems that may occur during a WRITE operation

Log Level	Information
ERROR	Serial port is off. Please turn on the serial port first
ERROR	No device managed
ERROR	Device {device} is not managed
ERROR	Offset {offset} for device {device} must be in range [0, {max_offset}]
ERROR	Writing problem in device {device} at offset {offset}
ERROR	Packet {packet} for device {device} is corrupted
INFO	Data written correctly

The minimum time required to assure that every device has performed a power cycle is approximately 30 seconds. For safety purposes, this station waits at least 1 minute. This pause is necessary to make sure that the SRAM contents are completely erased, knowing that an attacker could exploit data remanence and get PUF responses indirectly[1][2].

Moreover, a series of quality metrics common in PUF evaluation the quality of the samples has been studied. An in-depth analysis on the evaluation methodologies for PUFs is provided later. These quality metrics are measured when a new readout is performed and monitored through Grafana (https://grafana.com/) in real time to assess potential problems during the readout. Figures Figure 87 display two snapshots of the Grafana dashboard checking the quality of the samples. Indeed, a Grafana dashboard is used next to the devices to keep track in real time of any problem or any perturbation in the metrics.

Besides, fail-safe mechanism are also employed to detect other problems to immediately stop the platform to secure data integrity. (i.e. a command could hang indefinitely waiting for lost bytes or part of the devices could remain inaccessible due to a physical problem in the chain).

Current limitations

One of the main constraints of this platform is the acquisition time of the samples. The current setup follows a chain-like structure, the memory content is thus received and transmitted one after the other according to the position of the board in the chain. With 84 devices connected in one chain, it takes approximately 15 minutes to transmit the 80 kilobytes of the entire SRAM of the last board of the chain; therefore, the memory from all 84 devices would be retrieved in about 24 hours. In the event of a communication error, a packet is sent back to the PC and the information is retransmitted, resulting in additional transmission delays. Fortunately, these errors are extremely rare; we did not observe any communication failures after deployment.

The chain structure was chosen over a parallel structure to circumvent the physical limitations of the USB protocol, which supports a maximum of 128 devices, and to reduce the cost of powering and connecting each device (e.g., one USB per chain instead of one per device). Although certain enhancements could improve this setup, we believe it strikes a good balance between data integrity, scalability, and time.

An additional limitation constraining our analysis involves the absence of control over voltage and temperature parameters. Devices are exposed to environmental conditions and their inherent fluctuations. While this scenario provides valuable insights by simulating

[1] Y. Oren *et al.*, 'On the effectiveness of the remanence decay side-channel to clone memory-based pufs,' in *Cryptographic Hardware and Embedded Systems - CHES 2013*, G. Bertoni and J.-S. Coron, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 107–125, ISBN: 978-3-642-40349-1.

[2] M. Liu *et al.*, 'A data remanence based approach to generate 100% stable keys from an sram physical unclonable function,' in *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, IEEE, 2017, pp. 1–6.

real-world conditions, it restricts our ability to study device behaviour under specific critical conditions.

Code and Data Availability

A detailed description on the datasets and the platform has already been published in [4]. The source code for the platform and the STM32 devices along with the usage documentation is available under the GPL-2.0 licence at https://servinagrero.github.io/SRAMPlatform. A static version of the full dataset available at the time of writing can be downloaded from Zenodo [5] 10.5281/zenodo.7529512. The full dataset is composed of 2 CSV files that contain SRAM readouts of 84 Nucleo microcontrollers of STM32 type along with their voltage and temperature sensor data. Although the dataset described in this document is static, more data can be requested online through the following application hosted at https://puf4iot.univ-grenoble-alpes.fr/form.php. Any user can submit a request to the server specifying the data they want to retrieve. The server will process the request and return the data in a CSV file or in a zip file if the number of records is too large.

Measurements from industrial circuits

III.3

During the latter stages of this research, data from a circuit designed by Infineon was obtained. The circuit under investigation is a prototype of a large general-purpose SoC built for exploratory purposes using the latest technology design. It is designed for use in industrial environments under severe operational conditions. In addition to extensive areas of digital logic, the SoC includes a mixed-signal part, onboard memories, monitoring, and learning structures, as well as a substantial number of on-chip Ring Oscillators.

These ROs are grouped into six identical modules spatially distributed over the SoC, labelled A, B, C, D, E, and F. Each module contains 255 ROs with various path topologies. Some paths consist of homogeneous gates such as inverter gates, NOR gates, or NAND gates, while others are path replicas of dedicated paths in the design. All ROs oscillate in the range of 50 to 400 MHz under nominal conditions and are intended for timing and process monitoring purposes. Although not originally designed to function as PUFs, the identical design of the six modules allows for the possibility of an RO-based PUF. By comparing the frequencies of theoretically identical ROs in the same positions across two of the six modules, a PUF response can be generated.

Figure 7 provides a visual representation of the circuit. While the image does not represent the actual circuit due to its classified nature, it offers a rough estimate of the amount of data and the flexibility available for creating the PUF.

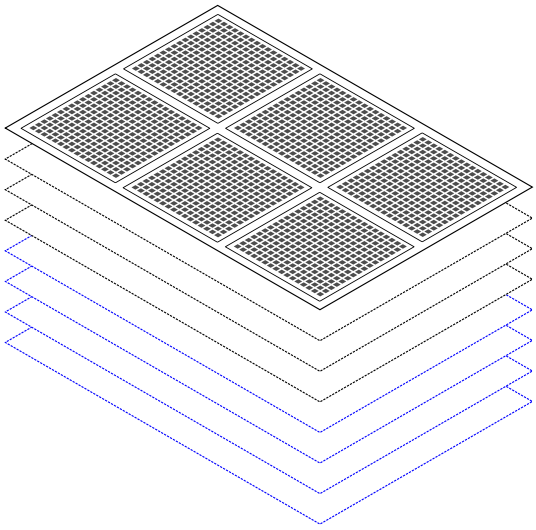


Figure 7: Visual representation of the industrial circuit. Each device has 6 blocks of 256 ROs. Each device has 8 measurements: 4 from the back end and 4 from the front end

The ROs are measured using an on-chip counter, which is the same procedure used in RO-PUFs. The RO frequencies are measured during the manufacturing stages at the wafer level (front-end, FE) and on the packed SoC in the back end (BE). Each frequency read-out is conducted at two temperatures (cold, hot) and two voltages (minimum, nominal). The frequency measurement is accurate, and the voltage and temperature are controlled for perfect conditions. The frequency values of all ROs are logged and used for this experiment. This results per SoC in 8 measured sets of $6 \times 255 = 1530$ RO frequencies along the manufacturing process.

The PUF design proposed in Figure 8 was conceived as follows: for each position i of an RO, there are 9 possible challenges, i.e., all combinations of blocks $(b1\ \ b2) = \{AD, AE, AF, BD, BE, BF, CD, CE, CF\}$, leading to overall 2295 CRPs.

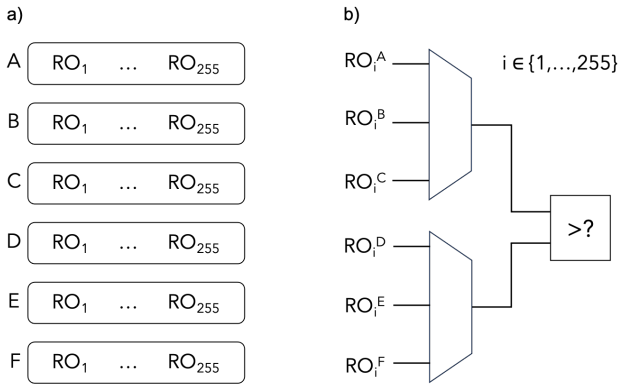


Figure 8: Schematic representation of the Infineon circuit under study: a) the six modules containing 255 ROs; b) the principle of operation of the PUF emulator

Ring Oscillator Simulated

III.4

Electrical simulations are of paramount importance in current electronics designs. First, simulations allow for the comprehensive analysis of the circuit behaviour under a variety of conditions without the need for extensive physical hardware, thereby saving time and resources. By modeling the electrical characteristics of ROs, simulations can predict how the PUF will respond to different environmental factors such as temperature variations, supply voltage fluctuations, and ageing effects. This predictive capability is essential for designing robust RO-PUFs that maintain high reliability and security across a wide range of operating conditions. Additionally, simulations enable the exploration of different design configurations and parameter settings, allowing researchers to optimize the performance of RO-PUFs in terms of uniqueness, stability, and randomness of the generated keys. By identifying potential vulnerabilities and performance bottlenecks through simulations, it is possible to make informed design improvements before moving to the costly and time-consuming process of physical prototyping and testing. In this section, the design of the RO-PUF under study and the simulation workflow used during this thesis is described in detail.

The RO-PUF under investigation comprises 200 ROs, divided into two groups of 100 ROs each (from $RO_{1,1}$, to $RO_{1,100}$ and $RO_{2,1}$, to $RO_{2,100}$) as shown in Figure 2.

To generate the complete set of Challenge Response Pairs shortplural (CRPs), the entire circuit must be simulated multiple times—once for each challenge, which for this configuration results in 10,000 simulations given the 100×100 possible combinations of ROs. An enable signal controls the oscillation. The chosen configuration for the RO includes a NAND gate and an even number of inverters, as depicted in Figure 9.

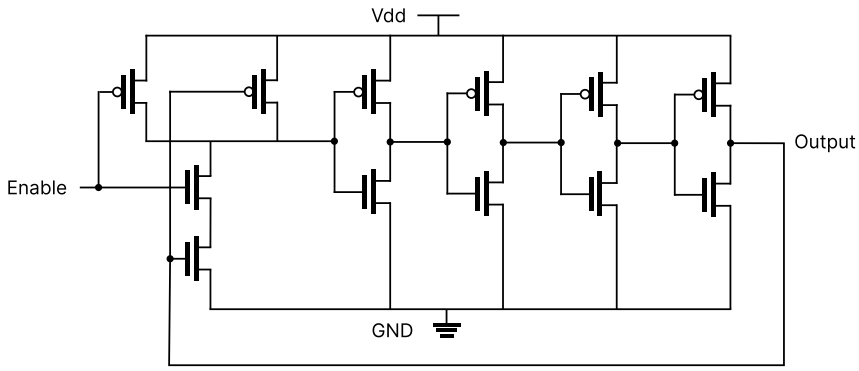


Figure 9: NAND RO CMOS Schematic

The output of each RO is connected to a counter (implemented in Verilog-A), which increments its value with each rising edge of the oscillation. Hysteresis behaviour is incorporated

in the counter to mitigate measurement errors. The source code for these modules is provided in the Appendix B.3.

To expedite simulation, each RO was simulated independently, and its frequency was indirectly measured by the counter, sampled every 1ns. The Ocean script for basic transient simulations is available in the Appendix B.1, and the template for Ocean to be used with Monaco is presented in B.2.

To emulate manufacturing process variability, random variations in the width, length, and threshold voltage (V_{th}) of each transistor were applied. While we can update many more parameters, these offer a very good compromise between accuracy and simulation speed. Additionally, simulations were conducted across different temperatures (24 to 30 ℃) and voltages (0.9 to 1.1V). Usually, a 10% range of variation for the supply voltage is agreed upon. The narrow range of temperature variation was chosen to focus on the gradient effects between two closely packed ROs on the same device. These parameter values assume minimal temperature gradients among ROs due to their proximity and potential voltage differences caused by power line resistance. The nominal values for all parameters are summarized in Table 10.

Table 10: Nominal values for the Ring Oscillator

Parameter	Nominal Value	Description
W_p	270 nm	PMOS transistor width
W_n	135 nm	NMOS transistor width
L_p	60 nm	PMOS transistor length
L_n	60 nm	NMOS transistor length
V_{thP}	0.45 V	PMOS transistor threshold voltage
V_{thN}	0.45 V	NMOS transistor threshold voltage
T	27 °C	Temperature
V_{DD}	1 V	Supply Voltage

The threshold voltage was modelled using a normal distribution, derived from the following computations based on[1]:

$$A_{\Delta VT,P} = 2.85e^{-9} \qquad A_{\Delta VT,N} = 3.46e^{-9}$$

$$\sigma_{V_{th}} = \frac{A_{\Delta VT}}{\sqrt{W \times L}}$$

[1] C. M. Mezzomo *et al.*, 'Characterization and modeling of transistor variability in advanced cmos technologies,' *IEEE Transactions on Electron Devices*, vol. 58, no. 8, pp. 2235–2248, 2011.

$$Vth \sim N(V_{th_0}, \sigma)$$

Each individual simulation last 512 ns with a fixed time step of 3 ns. The number of noise simulations was set to 11 per RO. That means that spectre will always run a nominal simulation, and then it will evaluate 11 different noise seeds on the same design.

Furthermore, noise was also included in the simulations, with values chosen according to[2][3][4]. The proper configuration of noise requires setting the variables `noise_fmin` and `noise_fmax` which are set to the complement of the simulation time and simulation step in nanoseconds respectively.

[2] R. Sarpeshkar *et al.*, 'White noise in mos transistors and resistors,' *IEEE Circuits and Devices Magazine*, vol. 9, no. 6, pp. 23–29, 1993.

[3] C. G. Theodorou *et al.*, 'Noise-induced dynamic variability in nano-scale cmos sram cells,' in *2016 46th European Solid-State Device Research Conference (ESSDERC)*, IEEE, 2016, pp. 256–259.

[4] C. G. Theodorou *et al.*, 'Dynamic variability in 14 nm fd-soi mosfets and transient simulation methodology,' *Solid-State Electronics*, vol. 111, pp. 100–103, 2015.

Conclusions

III.5

For a proper assessment of PUF designs, statistically significant data is essential, requiring access to numerous devices and multiple samples from each device. The scarcity of accessible datasets for PUF analysis motivated the development of an open-source and open-access platform named SRAMPlatform. This platform was designed to collect extensive SRAM data and sensor readings from microcontrollers. Currently, the platform gathers data from 84 STM32 microcontrollers, storing the data weekly in an open access database, available for use by the research community, facilitating broad research opportunities. This platform allows the execution of external code to study certain phenomena directly on the device. One phenomenon of interest is NBTI, which is crucial for examining ageing effects and devising techniques to enhance the entropy and reliability of PUFs.

Towards the end of the thesis, access to real Ring Oscillator data from Infineon was granted. This comprehensive dataset comprises 399 devices, each with 6 blocks of 256 ROs. From each device, we have 8 measurements: 4 from the Backend and 4 from the Frontend. Although this dataset was not initially intended for ROs-PUFs, it provided an invaluable resource for validating our simulation hypotheses and exploring novel techniques and designs to enhance the performance of RO-based PUFs. Both datasets allowed for significant advancements in both the validation of our models and the innovation of new approaches in PUF research.

A summary of the different PUF designs under evalution is presented in Table 11 and a detail analysis of the data gathered is presented in Chapter III.3.

Table 11: Summary of the tested PUF designs

PUF Circuit	Number of circuits	CRPs	Operating Conditions	Uses
Simulated RO-PUF	200 ROs	10000	24 °C to 29 °C, 0.9 V to 1.1 V	Evaluation of Reliability and Bit-aliasing in Chapter III.2 and development of filtering technique in Chapter III.2.
SRAM Platform	84 boards	655360	Environmental conditions of approximately 27 °C and 3.3 V. One control group of 42 boards and another NBTI testing group of 42 boards	Metric evaluation and validation of correlation between metrics in Chapter III.3. Mathematical modelling in Chapter III.2.
Industrial-grade RO	399 devices with 6 blocks of 255 ROs each	2295	Not provided	Validation of correlation between metrics in Chapter III.3 and Split PUF design in Chapter III.1

IV

PUF Canonical Evaluation Metrics

PUF Canonical Evaluation Metrics

IV.1

Analogous to cryptographic functions and other security primitives, a mathematical model describing PUF behaviour is essential for validating security claims. This mathematical abstraction not only facilitates security validations but also enables comparisons between various designs. Since the conception of PUFs, different evaluation frameworks have been proposed and extended to characterize their behaviour.

Maiti et al.[1][2] proposed performance metrics that are nowadays considered canonical, namely Uniformity, Bitaliasing, Inter-Hamming Distance (i.e., Uniqueness), and Intra-Hamming Distance (i.e., Reliability). The formulas presented in this section are extracted directly from their work. These canonical metrics utilize Hamming Distance between responses and Hamming Weight to measure the properties of the PUF for each individual device and the overall behaviour across devices, as summarized in Table 12.

Table 12: Metrics classification according to methodology

	Same Device	Different Devices
Hamming Weight	**Uniformity**	**Bit-aliasing**
Hamming Distance	**Reliability**	**Uniqueness**

An in-depth mathematical explanation about the ideal values of these metrics is found in [144, 145], and it is tightly related to Shannon Entropy and Guesswork, which is the amount of effort an attacker needs to guess a secret.

1.1 Mathematical representation

A small visual representation is provided to better understand how the metrics are computed. This foundational understanding is crucial for accurately interpreting the metrics and their implications for PUF performance. Since there's not a standardized method for storing the CRPs and evaluating the metrics, here it's presented the representation that is used consistently throughout this document.

First, a series of definitions that will be used to refer to the different aspects of a system:

[1] A. Maiti *et al.*, 'A large scale characterization of ro-puf,' in *2010 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, IEEE, 2010, pp. 94–99.

[2] A. Maiti *et al.*, 'A systematic method to evaluate and compare the performance of physical unclonable functions,' *Embedded systems design with FPGAs*, pp. 245–267, 2013.

- A PUF *instance* refers to the actual physical circuit that works as the PUF. The words *instance* and *device* are equivalent.
- A *sample* refers to a complete measurement of all challenges of an device in time. Subsequently, the reference sample is the set of responses used as the reference to compare all succeding responses.
- S refers to the set of all CRP samples of the system. $S_{d,c,s}$ refers to the sample s of the CRP c for device d.
- C refers to the set of all CRPs of the PUF. $C_{d,c}$ refers to the CRP c for device d.
- D refers to the set of all devices in the system. D_n refers to device number n.

On a numerical representation, the CRPs are stored in multi-dimensional arrays. Each matrix represents a sample, where each row corresponds to a device and each column to a challenge. The 3rd dimension of the array represents the samples as depicted in Figure 10.

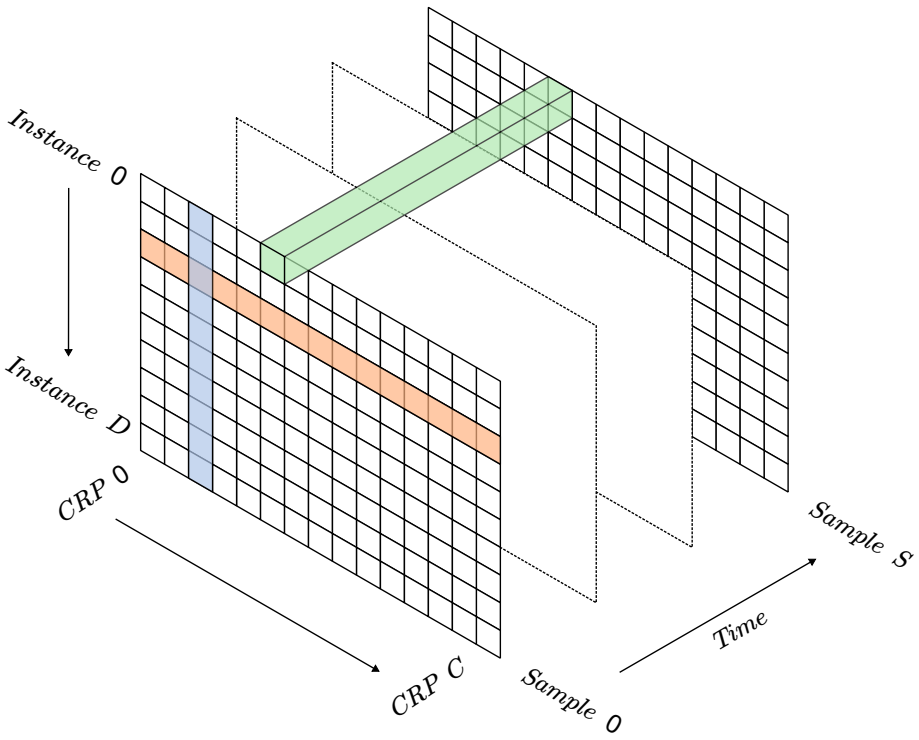


Figure 10: Visual representation of the arrangement of CRPs and computation of PUF metrics. Bit-aliasing is represented in blue; Uniformity in orange; Reliability in Green and Uniqueness is computed on pairs of instances.

1.2 Uniformity

Uniformity measures the ratio of 1s and 0s across all responses for each PUF instance. An uneven distribution of 1s and 0s in the PUF responses may allow an attacker to extrapolate some response given they have access to a subset of the responses. It is computed as depicted in Equation 1.1, and its ideal value should be 50%, which indicates that a device produces an even distribution of 1s and 0s.

$$\text{Uniformity}_i = \frac{1}{n}\sum_{l=1}^{n} r_{i,l} \times 100\% \tag{1.1}$$

where $r_{i,l}$ is the l–th binary bit of an n–bit response from a chip i.

1.3 Bit-aliasing

Bit-aliasing measures the ratio of 1s and 0s across responses for each challenge. Akin to Uniformity, an uneven distribution of 1s in the 0s in the PUF responses may allow an attacker to extrapolate some of the responses. Moreover, an uneven distribution decreases the Entropy of that challenge, that is, it reduces the ability of the PUF of uniquely identifying devices.

It is computed as depicted in Equation 1.1, and its ideal value should be 50%, which indicates the even distribution of values. As it can be seen by comparing Equation 1.1 and Equation 1.2, Bit-aliasing shares many similarities with Uniformity as they measure the same concept in different dimensions (e.g., average response value per challenge instead of per device).

$$\text{Bit-Aliasing}_l = \frac{1}{k}\sum_{i=1}^{k} r_{i,l} \times 100\% \tag{1.2}$$

where $r_{i,l}$ is the l–th binary bit of an n–bit response from a chip i. Akin to Uniformity, the ideal value is 50% which indicates an even distribution of values across devices.

1.4 Uniqueness

Uniqueness represents the ability of a PUF to uniquely distinguish a particular chip among a group of chips of the same type.

$$\text{Uniqueness} = \frac{1}{\text{Number of Pairs}}\sum_{i=1}^{k-1}\sum_{j=i+1}^{k}\frac{HD(R_i, R_j)}{n} \times 100\%$$

The scaling factor for the Uniqueness is given by the total number of pairs of devices in the system, and can be computed as shown in Equation 1.3.

$$\text{Number of Pairs} = \binom{k}{2} = \frac{k\,(k-1)}{2} \tag{1.3}$$

It's important to state as it can be derived from Equation 1.3 that the number of comparisons performed grows rapidly, as every new device needs to be compared against all the devices in the system.

1.5 Reliability

The Intra Hamming-Distance, although normally it's complementary value Reliability is used instead, measures how efficient a PUF is in reproducing the response bits. The Intra Hamming-Distance measures the deviation (i.e. distance) of a response from the reference one. For this, an initial reference sample R_i is first extracted at normal operating conditions. In order to evaluate then deviation, multiple responses are subsequently extracted at different operating conditions and they are compared to the reference sample as shown in Equation 1.4. The ideal value of Intra Hamming-Distance should be 0% which indicates no deviation from the reference sample.

$$HD_{INTRA} = \frac{1}{m}\sum_{t=1}^{m}\frac{HD(R_i, R'_{i,t})}{n} \times 100\% \tag{1.4}$$

Finally, the Reliability is computed as the complement of the Intra Hamming-Distance as shown in Equation 1.5, so subsequently, it's ideal value is 100%.

$$\text{Reliability} = 100\% - HD_{INTRA} \tag{1.5}$$

Further study is needed to elucidate reasonable environmental conditions under which to evaluate the PUF, specially taking into consideration how difficult it is to properly quantize the effect of noise, external factors and even intrinsic characteristics of each PUF design. For example, in the context of RO-PUFs the authors in[3] found that the accuracy of the measurements performed on Silicon PUFs are limited by environmental noise, even for large measurement times. As such, the authors claim that it is suboptimal to choose long measurement times to try to achieve accurate measurements. However, this still remains a problem under several evaluation proposals.

[3] R. Maes *et al.*, *Statistical Analysis of Silicon PUF Responses for Device Identification.* Germany: Berlin, 2008.

1.6 Additional proposals

The authors in [29] proposed their own set of metrics, namely Predictability, Collision, Sensitivity, and Susceptibility as means to test the resilience of the PUF to attacks: Predictability evaluates the difficulty of predicting the PUF responses given the corresponding challenges; Collision estimates the likelihood of two different PUFs yielding the same responses to the same given inputs; The Sensitivity parameter ensures that the amount of manufacturing variability is large enough such that a PUF operates in a stable way when the components are imperfect; and lastly, Susceptibility, determines the hardness of characterizing the PUF circuit components. They provided statistical parameters with single-bit granularity, which is beneficial for assessing PUF responses as a whole. This idea serves as the basis for more complex analyses tailored to different PUF families.

The authors in [146] proposed around the same time as Maiti their own metrics, including Randomness, Uniqueness, Correctness, Diffuseness, and Steadiness. Maiti explained that the definitions of Randomness, Correctness, and Uniqueness are slightly different from the canonical Uniformity, Reliability, and Uniqueness respectively, although they measure similar aspects. Despite the similarities, Maiti argues that during the evaluation of the Steadiness, which measures the degree of bias of a response bit towards '0' or '1' over T samples, the time stamps of the sample measurements are important since the steadiness of a PUF may change when operating conditions change. However, Hori et al. did not discuss the effect of time on the steadiness parameter. Maiti also highlight another problem regarding Diffuseness. While it resembles the computation of Uniqueness, Diffuseness is defined inside a single chip among several IDs. Since this value is estimated among K signatures of L bits each generated in a chip, these $K \times L$ bits can be divided in many possible K groups with L bits each and the same set of $K \times L$ bits can lead to different results based on the combination selected. This problem is aggravated due to the fact that the challenges are selected by a software program, leading to deterministic results.

The works in [147, 148] provide in-depth performance evaluations of delay PUFs, suggesting that tests can be tailored specifically for the idiosyncrasies of different PUF designs.

The analysis of PUF metrics has also attracted attention from the cryptography and mathematics communities. For instance, [149] attempted to formalize PUF metrics holistically but analogous to the canonical metrics, there is no distinction between strong and weak PUFs. In [150], a theoretical framework linking PUFs with the Random Oracle is provided. A random oracle responds to every unique query with a uniformly random response.

The authors in [151, 152] provide comprehensive information on developing fuzzy extractors to generate secrets from PUF designs, closer to signal processing and reliability enhancement rather than PUF evaluation.

Another important work is showcased in [153], where the authors provide an all-in-one assessment methodology for delay-based PUFs, considering external factors like side-channel and modeling attacks.

In [154, 8], the canonical metrics are extended, offering a methodology to estimate the Reliability and Entropy of different delay PUFs revealing an Entropy-Reliability trade-off. This relationship is studied extensively in the next chapter.

In [155], it is emphasized that the confidence interval of bit-aliasing is often overlooked, leading to an overestimation of a PUF's unpredictability. The importance of a substantial sample size for accurate entropy assessment is underscored.

Other extensions have been proposed, such as in [156], where the authors propose calculating entropy and joint entropy to highlight design issues, and [157], where current metrics are extended to provide spatial analysis of all responses. The following section will delve into this type of topological and spatial analysis.

Efforts to unify metrics can be found in [158], where the authors merge previously proposed metrics into a single set to avoid redundancy and create consistent tests.

The metrics proposed by Maiti have remained the canonical ones since and although not perfect, they capture the critical aspects in a simple and concise manner. However, the canonical metrics along with a multitude of additional proposals share a series of limitations. While metrics have a robust mathematical basis, others are arbitrary, redundant, fail to provide information to discriminate PUFs based on security requirements or fail to provide information back to designers in order to improve the PUF performance. Furthermore, issues related to their computation and representation can impact their effectiveness and applicability in certain contexts. In the following chapter, the limitations of the standard evaluation metrics are explored in detail and a series of mitigations and extensions are proposed to enhance the robustness and accuracy of PUF evaluation metrics.

Indeed, several studies found in the existing literature express the significance of a large device population to test PUF characteristics, cautioning against relying solely on average quality metric values. Novel studies like the one presented in[1] in 2023 bring attention to the hidden problems associated with presenting solely the mean uniqueness value. Relying solely on the average is an inadequate metric, as it may yield inaccurate results when confronted with outliers or complementary patterns, that is to say, the mean is not a sufficient statistic of a distribution. This is also aggravated from the fact that the derivation of metrics from already established statistical parameters is non-trivial as it can lead to masking issues as shown in[2]. For instance, in many metrics, the ideal range falls between 0 and 1, with 0.5 being the optimal value. However, a skewed distribution, concentrated between 0.25 and 0.75, can result in an average of 0.5. While the standard deviation addresses this asymmetry, it is often overlooked. Figure 11 showcase a series of synthetic cases where the mean value of the metrics is around the ideal value while clearly exhibiting extreme biasing. Furthermore, later in this document real experimental data from the SRAM-PUF proves these claims, since the heave Bit-aliasing effects are masked in the standard deviation.

Moreover, due to cost constrains, the evaluation of PUFs is always performed on a test data set, which is never the same size as the final population. Moreover, it's difficult to ensure good sampling as devices would need to be drawn from different manufacturing lots to ensure proper sampling. Because of these constrains, the metrics should be computed along with Confidence Intervals as demonstrated in[3][4]. Thus, as explained in [158], that tests can be seen as estimators from their theoretic expressions, the so-called "stochastic models" as the authors call it. Since in most situations evaluating the PUF on all the device population is unfeasible, we propose later in this document a statistical methodology to extrapolate the evaluation results given the same dataset.

Many of the studies in the literature are conducted on a limited chip population or older technology nodes, potentially concealing crucial issues relevant to the PUF behaviour. Moreover, situations may arise where design issues or manufacture problems can limit the PUF entropy. Yet, these issues can be overlooked if the remaining responses provide enough randomness or whose average values are close to ideal. This problem grows larger with the number of available CRPs, making it challenging to comprehensively analyse each response. Beyond highlighting entropy-related errors, a thorough analysis of metrics

[1] Y. Wei *et al.*, 'Apuf production line faults: Uniqueness and testing,' in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2023, pp. 1–6.

[2] C. Frisch and M. Pehl, 'Beware of the bias-statistical performance evaluation of higher-order alphabet pufs,' in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2022, pp. 1005–1010.

[3] F. Wilde and M. Pehl, 'On the confidence in bit-alias measurement of physical unclonable functions,' in *2019 17th IEEE International New Circuits and Systems Conference (NEWCAS)*, IEEE, 2019, pp. 1–4.

[4] F. K. Wilde, 'Metrics for physical unclonable functions,' Ph.D. dissertation, Universität München, 2021.

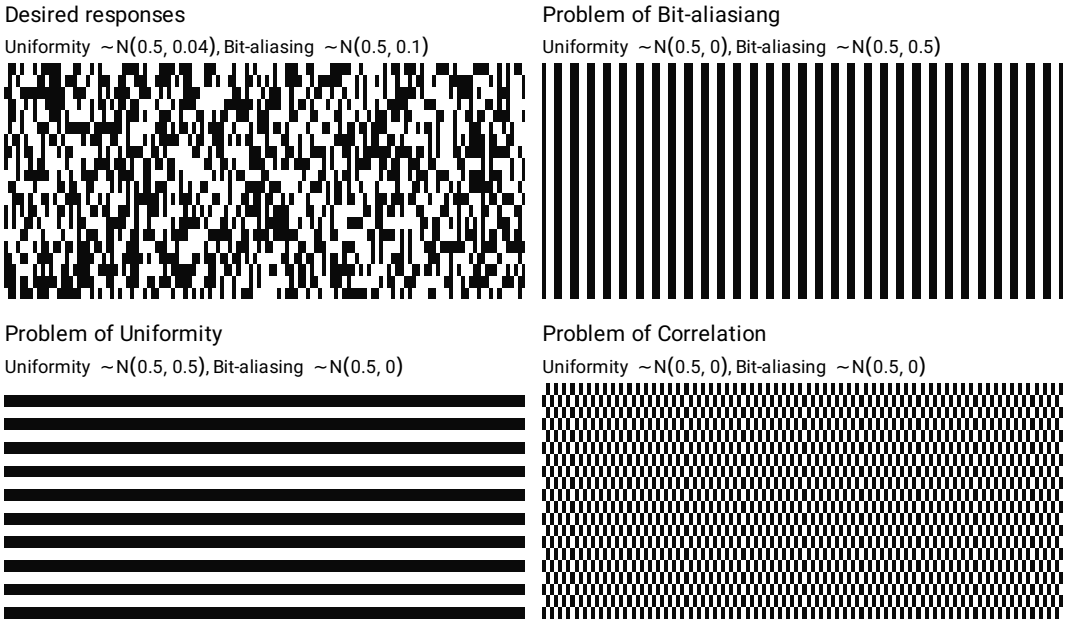


Figure 11: Desired set of CPRs and examples showcasing different phenomena

can offer indispensable insights for PUF designers. Notably, there is a scarcity of studies addressing PUF entropy analysis or the impact of manufacturing variations on entropy generation.

2.1 Lack of correlation analysis

A critical aspect often overlooked in PUF response analysis is the study of correlation, which ensures that all responses are independent of each other. Correlations not only reduce the entropy of the PUF and deteriorate its performance, but they also provide an avenue for attackers to exploit these dependencies for targeted guessing. Ganji et al. [161] demonstrated that PUF responses cannot be considered as a set of uncorrelated bitstrings. Instead, a subset of bit positions, referred to as influential bits, exhibit truly random behaviour while other bits are correlated with these influential bits. Other studies [157, 162, 163] have highlighted the effectiveness of spatial analysis, which examines the topological or geometric properties of PUFs.

Furthermore, studies such as [164] highlight the significance of spatial phenomena in generating sufficient entropy. These studies experimentally demonstrate that due to systematic fabrication variations, common topologies of RO-PUFs often fail to pass the NIST randomness test suite, which is typically used to evaluate cryptographic functions. To address this issue, they propose a solution involving entropy distillers based on polynomial regression to enhance the randomness and reliability of the RO-PUFs. A similar approach is presented in [165], where the authors argue that systematic process variation is the

primary cause of failures in randomness tests, such as those by NIST, for RO-PUFs. In their work, the authors propose modeling these variations and retaining only the truly random component of the process variation to improve the design of the RO-PUF.

Spatial correlation analysis have been used to conduct successful attacks. For example, Bahar Talukder et al. [166] experimentally demonstrated that signatures from two memory chips could exhibit highly correlated properties if they share the same specifications and manufacturing facility. Furthermore, the analysis of 1000 Arbiter PUF-based RFID tags by Utz et al. [167] highlighted that simple Uniqueness and Reliability analysis might not reveal issues such as implementation biases or defects causing deviations from standard behaviour.

To clearly demonstrate the aforementioned effects, consider a PUF where each device yields a 128-bit response. The Uniformity of every single device is 0.5, but due to a design issue, the second half of the 128 bits is a complementary mirror image of the first half as showcased in the heatmap in Figure 12. This example illustrates how canonical metrics may fail to identify correlation or interdependence effects.

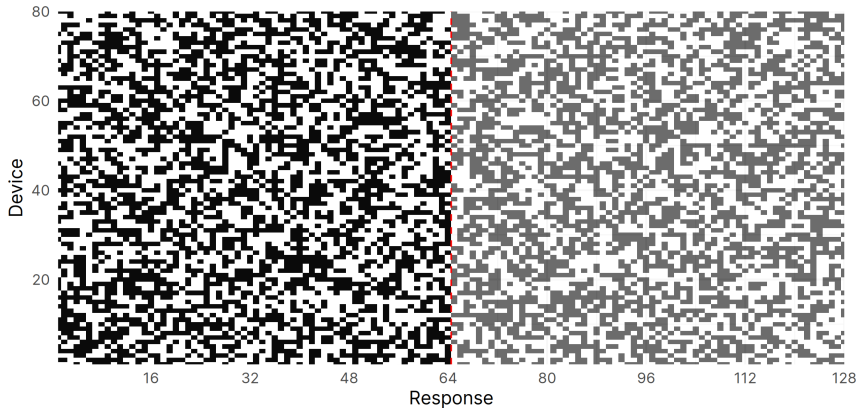


Figure 12: Heatmap of highly correlated CRPs

As it can be seen in Figure 13, the distribution of Uniqueness for the ideal set of CRPs and the set of CRPs with heavy correlation effects are practically identical. The numerical differences are not sufficiently large to discern these kinds of correlation effects.

Failing to highlight these interdependencies can have serious security implications, as attackers can exploit these effects to guess a large subset of a response given access to a small subset of response bits. Similar to Uniformity and Bit-aliasing, metrics that measure the distribution of response values within the same device and across different devices (as shown in Table 12), the study of interdependency and correlation effects is proposed later in this chapter in Section 3.1.1.

Additionally, for PUFs that generate bits sequentially, it is critical to perform temporal auto-correlation studies on the output to ensure that the generated bitstream is "memory-less," meaning it does not depend on past values. There are numerous methods to study

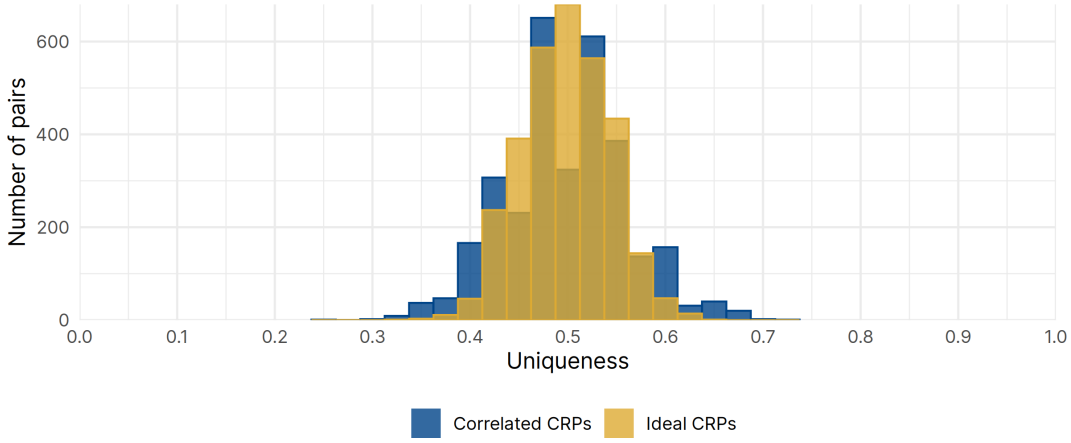


Figure 13: Uniquess comparison between ideal case and correlated case

these effects. The subsequent section proposes a series of methodologies to evaluate interdependency effects effectively.

2.2 System capacity and collisions

An aspect of PUF analysis that is rarely studied and is related to Uniqueness is what we call term *capacity*. We define capacity as the maximum number of responses that is physically possible to identify while mantaining ideal PUF properties, specificially a Uniformity of 50% for all devices and Bit-aliasing of 50% for all challenges. Parallel to this analysis, is the one of collisions, which occur when 2 different devices produce exactly the same key.

For a given bitstring of size n, the total number of unique combinations, referred to here as "hard capacity," is defined by:

$$\text{Capacity}_{Hard} = 2^n \tag{2.1}$$

To preserve a Uniformity of 50% across all devices and a Bit-aliasing of 50% across all challenges, only responses with an even distribution of bits are allowed. The number of unique combinations with exactly an even distribution (i.e., 50% of the bits are 1s and 50% are 0s), termed "soft capacity," is given by:

$$\text{Capacity}_{Soft} = \binom{n}{n/2} \tag{2.2}$$

In the context of soft capacity, an ideal response would exhibit an equal ratio of 1s and 0s. However, we can accommodate scenarios where this ratio is slightly biased. Specifically, we consider a bias of x bits from a perfectly balanced 50% distribution of 1s and 0s. For

Table 13: Hard and Soft capacity of a PUF given the response size.

n	Soft Capacity	Hard Capacity	Ratio
2	2	4	0.50
4	6	16	0.38
8	70	256	0.27
16	1e+04	7e+04	0.20
32	6e+08	4e+09	0.14
64	2e+18	2e+19	0.10
128	2e+37	3e+38	0.07
256	6e+75	1e+77	0.05
512	5e+152	1e+154	0.04

instance, a bias of 4 bits in a 64-bit response would result in a distribution where the soft capacity is not defined as 32/32, but rather as 28/36 or 36/28. By using Pascal's triangle, it is possible to derive the soft capacity for any given a deviation of x bits.

$$\text{Capacity}_{Soft} = \binom{n}{n/2} + 2 \times \sum_{i=n/2-x}^{n/2-1} \binom{n}{i}$$

The Soft capacity is equal to the Hard capacity if the deviation is exactly $n/2$.

$$\sum_{i=0}^{n} \binom{n}{i} = 2^n$$

It's evident from Table 13 that while the capacity is extremely large for more than 64 bits of response, (just for reference, it's estimated that there are 7×10^{22} stars in the known universe), the soft capacity, represents only a small fraction of the total capacity, and the ratio between the soft capacity and the hard capacity grows smaller with response size. Therefore, the capacity for real-life PUFs is not a limitation.

Collisions pose a threat as they can reduce the capacity. As previously said, the authors in [29] provide the Collision test to measure the likelihood of two devices giving the same response with the same inputs. Even in an ideal system, this is bound to happen due to the finite number of possible responses and the fact that eventually, all possible responses are bound to occur. Collisions may pose a problem as two devices would generate the same key and be wrongly identified so an attacker could exploit collisions to create clones of the same device.

The probability of collision P of at least one collision among k samples from N possible values can be approximated using the birthday paradox [168, 169].

$$P \approx 1 - \exp\left[-\frac{k(k-1)}{2 \cdot \text{Capacity}}\right]$$

For a general collision probability p, the approximate number of responses k can be computed as shown in Equation 2.3

$$k \approx \sqrt{\text{Capacity} \times \ln\left(\frac{1}{1-p}\right)} \tag{2.3}$$

The computation can be summarised to the following assuming a 50% chance of a collision:

$$k \approx 1.177\sqrt{\text{Capacity}} \tag{2.4}$$

While it's clearly demonstrated that the probability of collision remains extremely low across a diverse number of devices, it's important to state that these values represent the upper bound, (e.g. best case scenario), where the PUF responses are completely random and an attacker cannot exploit any effects to their advantage. The probability of collision increases due to Bit-aliasing and correlation effects.

2.3 Lack of a reference model

As demonstrated in the preceding sections, metrics are typically represented by their mean values, often neglecting the spread of these distributions. However, even when the spread is provided, there is a significant gap in understanding regarding the permissible ranges or the relationship between these spreads and system characteristics, such as the number of devices or the number of responses.

To the best of our knowledge, there is no definitive model that accurately describes the ideal distributions for different PUF designs under varying system characteristics. This absence poses a critical problem: the current metrics cannot discriminate different PUF designs based on different security requirements. This issue is well-documented, as evidenced in [170, 171], who demonstrate that the Uniqueness distribution shifts significantly with the number of devices in the system. It is thus critical to have a golden model that can be used as reference no matter the extent of the system under study. This issue will be addressed in later in Chapter IV.2.

Proposed mitigations and extensions

IV.3

In the following sections, a series of mitigations to the aforementioned issues are proposed. Besides a series of new metrics are also proposed as a way to provide more information to designers when performing PUF analysis, based on what has been said that metrics fail to discriminate certain PUF designs give some security requirements.

3.0.1 Deviation from ideal value

As a means to mitigate the problems related to omitting the standard deviation in the representation of the metrics, the authors in [159] advocate to use the deviation of the metrics from the reference value (i.e. 0.5) instead of the metrics themselves. While there exist a plethora of functions that fit this description, the simplest one the authors propose is the Manhattan distance, commonly used in regression analysis, defined as

$$g(p) = |p - p_{ideal}| = |p - 0.5|$$

While this simple option is efficient, there are other formulas that are, in nature, tied to the field of information and randomness and could provide more insight about the PUF performance than the simple Manhattan Distance. The only requirement for these functions is to be monotonic in order to remove the masking issues that arise when displaying only the average value. Among the series of functions that fit the described criteria, we propose the following:

Kullback-Leibler Divergence

The Kullback-Leibler Divergence denoted as $D_{KL}(P, Q)$ measures the difference between a probability distribution P and a reference probability distribution Q. This function is defined in detail in Equation A.19.

$$g(p) = D_{KL}\left(p, Q(x)\right)$$

Where $Q(0) = Q(1) = 0.5$ which corresponds to an "ideal" Binomial distribution.

Shannon Entropy

The Shannon Entropy of a probability distribution, whose computation is shown in Equation A.18 is another valid function. For binary responses it can be summarised as follows:

$$g(p) = \mathrm{H}(p) = -\left[p \log_2(p) + (1-p) \log_2(1-p)\right]$$

Bit Ratio

This function can be used as a direct replacement for Uniformity and Bit-aliasing as it provides information about the ratio of 1s and 0s in a bitstring. The absolute value of the bit ratio should equal the Uniformity of a bitstring.

$$BR = \frac{|\text{Number of 1s} - \text{Number of 0s}|}{\text{Number of bits}}$$

3.0.2 Stability

The computation of Reliability requires the measurement of a reference sample, a process known as enrollment, for the subsequent samples to be compared against. To obtain the the enrolment sample, many measurements are performed at different conditions and the most likely response is chosen. However, if the response to be analysed is slightly unstable in nature and this process is not followed correctly, it would be possible to obtain as a response the least probable outcome. That in turn would consider the response as unreliable as most of the ensuing responses will be the most probable outcome.

As a way to mitigate this issue, we propose Stability, which measures the maximum number of equal bits in a bitstring. Stability can be thought of as "Soft Reliability" because it is independent of the sample order, that is, it measures how resilient a response is to different environments and time. Stability is practical from a usage standpoint, as it provides only the most stable value without indicating when instabilities occur. It is computed as the Shannon Entropy of the single bit-response to the same challenge at different interrogations of the PUF as shown in Equation 3.1.

For example, given the set of responses $S = 011110$ that contains the responses of the same challenges after 6 interrogations, 2 out of the 6 bits are 0, which means that 1 is more stable. For this specific case, the Reliability will be $1/5 = 0.2$ as only the last response is identical to the reference one, while the Stability is $H(4/6) = 0.91$.

$$\text{Stability}(S) = \text{H}(S) \tag{3.1}$$

3.1 Reliability Invariance

Significant efforts have been made in the literature to create frameworks for studying Reliability, as highlighted in [172]. The standard evaluation of Reliability involves comparing the current response to the reference using the Hamming distance. While this approach is practical from an application standpoint, it falls short for in-depth analysis, as it loses critical information about individual bits. This lack of data could hinder hardware designers' efforts. To address this issue, we propose an extension to the current Reliability framework to study the "invariance" of Reliability against different phenomena.

Firstly, Reliability should be computed for each individual response as already proposed by Maes et al. This approach not only provides valuable information to designers but also sheds light on spatial phenomena. This is represented as shown in Equation 3.2.

$$\text{Reliability}(d, c) = \frac{1}{S} \sum_{s=2}^{S} R_{dc} \oplus R_{dc} \tag{3.2}$$

The canonical computation of Reliability results in averageing out the reliability of each individual bit.

$$\text{Reliability}(d) = \frac{1}{C} \sum_{c \in C} \left[\frac{1}{S} \sum_{s=2}^{S} R_{dc} \oplus R_{dc} \right] \tag{3.3}$$

Analysing the reliability of each device along with manufacturing metadata can help detect anomalies. We further propose extending this analysis by considering conditional values based on the reference sample, as shown in (3.4).

$$\text{Reliability}(c) = \begin{bmatrix} \text{Reliability}(c \mid s_0 = 0) \\ \text{Reliability}(c \mid s_0 = 1) \end{bmatrix} \tag{3.4}$$

This can be expanded into a "Reliability Invariance Matrix" by tallying all possible bit-flip transitions. Normalizing these counts yields the probabilities of different transitions, as represented in (3.7).

```
bitstring = [0 1 1 0 1 0 1 0]
N = length(bitstring)
RIM = np.zeros((2, 2))

for i in range(1, N - 1):
  x = bitstring[i]
  y = bitstring[i+1]
  RIM[x, y] += 1

RIM = RIM / (N - 1)
```

Listing 3.1: Computation of the Reliability Invariance Matrix in Python

$$\text{Reliability Invariance} = RI = \tag{3.5}$$

$$\begin{bmatrix} P(s_{t+1} = 0 \mid s_t = 0) & P(s_{t+1} = 1 \mid s_t = 0) \\ P(s_{t+1} = 0 \mid s_t = 1) & P(s_{t+1} = 1 \mid s_t = 1) \end{bmatrix} = \tag{3.6}$$

$$\begin{bmatrix} p_{00} & p_{01} \\ p_{10} & p_{11} \end{bmatrix} \tag{3.7}$$

While this matrix could theoretically be expanded to more than two states, the added computation would not yield additional useful information, especially since this method is sensitive to the sample order. Different sample arrangements can produce significantly different Reliability Invariance values.

Ideally, Reliability values correspond to both 0s and 1s being reliable (e.g., $p_{00} = p_{11} = 1$), resulting in the identity matrix. In Markov Analysis terms, the ideal system means that both the states 0 and 1 are absorbing states, where once the system enters that state, it remains there. The fact that the normalized matrix is a right stochastic matrix of transition probabilities, led to the creation of mathematical models based on Markov Chains for the analysis of PUF designs as it will be presented in Chapter IV.6.

$$RI_{Ideal} = I_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

As it will be shown in Chapter IV.3, the computation of the Reliability Invariance Matrix can highlight certain phenomena of interest, which are specially interesting for the SRAM-PUF studied here. This Reliability Invariance Matrix can be used on any kind of PUF. Moreover, it could be tailored to specific PUF designs by analytical studies like the ones performed in [173]

3.1.1 Correlation studies

Studying correlation effects is critical for fully assessing the threats posed by certain PUF designs. Here, we propose an extension to the auto-correlation matrix commonly used in digital analysis.

This extension is a preliminary step for detecting correlation effects. If such effects are detected, more advanced statistical analyses would be required. The proposed Auto-Correlation (AC) Matrix is computed by tallying the results of performing XNOR on every pair of individual responses. Due to the nature of the XNOR operation, the resulting matrix is symmetric. Its computation is shown in Equation (3.9).

$$(AC)_{xy} = (AC^T)_{xy} = XNOR(r_x, r_y) = r_x \otimes r_y \quad \forall x, y \in C \tag{3.8}$$

$$AC_{xy} = 1 \iff x = y \tag{3.9}$$

The example shown in Figure 14 represents the AC Matrix of a PUF biased towards 0. It's evident that since the example PUF yields a larger amount of 0s, there are a lot of correlations between different responses.

This autocorrelation matrix can easily be computed numerically and can be used to compute the joint probabilities of each pair of bits. The simplest way to account for the correlation

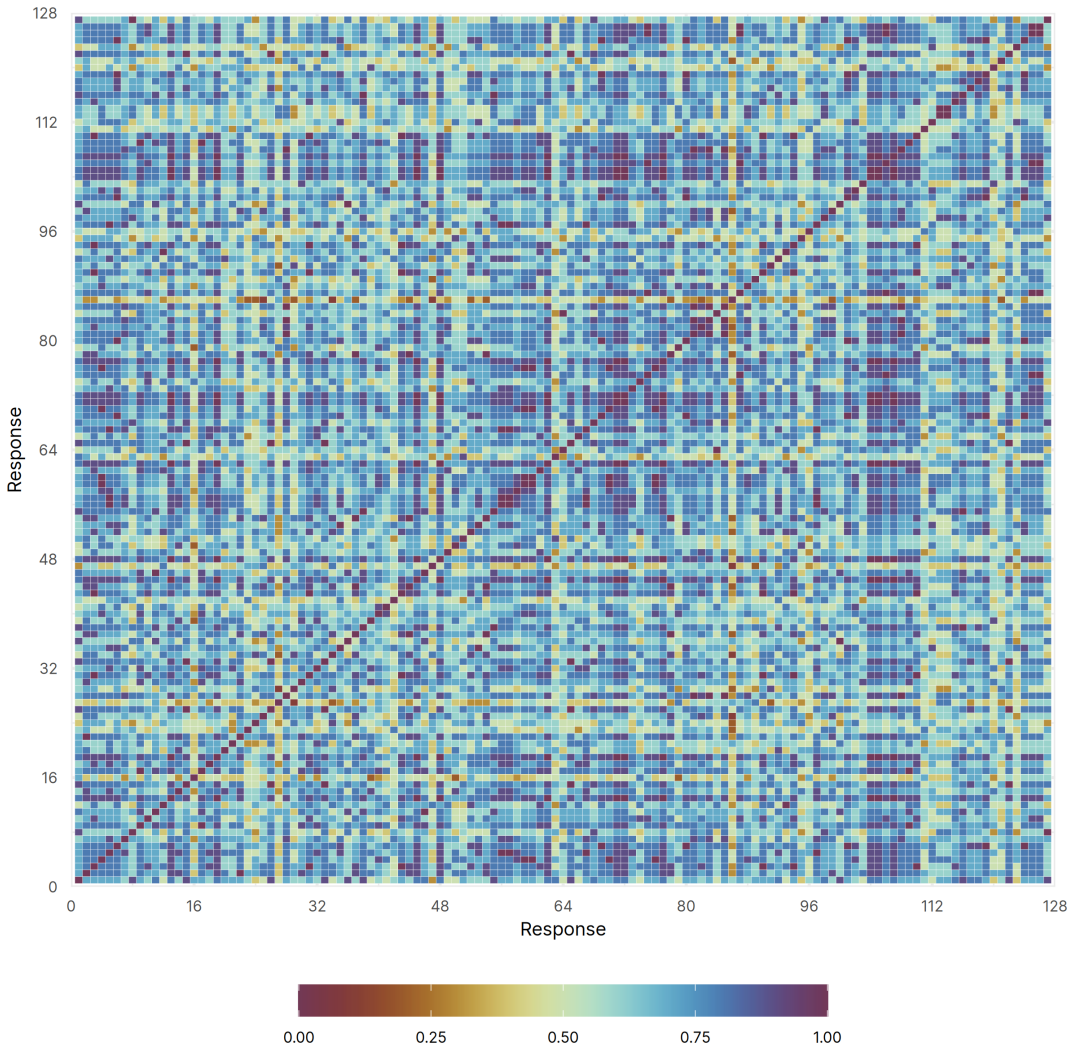


Figure 14: Auto-correlation matrix example of a PUF with Bit-aliasing=0.2

between two responses a and b is to compute the joint probabilities shown in Equation 3.10 based on the marginal probabilities (i.e. Bit-aliasing) of each response and the conditional probability $P(a = 0, b = 0) + P(a = 1, b = 1)$ given by the AC matrix.

$$\begin{bmatrix} P(a=0,b=0) & P(a=1,b=0) \\ P(a=0,b=1) & P(a=1,b=1) \end{bmatrix} \tag{3.10}$$

The final expression of the joint probabilities of a pair of responses a and b is shown in (3.11), where p_a, p_b refer to the marginal probabilities of a and b respectively and $AC_{a,b}$ refers to the probability of them being equal The final computations of the joint probabilities are depicted in, where p_a refers to the marginal probability of the response a.

$$\frac{1}{2}\begin{bmatrix} (AC_{a,b} - p_a - p_b + 1) & (p_a - p_b - AC_{a,b} + 1) \\ (p_b - p_a - AC_{a,b} + 1) & (p_a + p_b + AC_{a,b} - 1) \end{bmatrix} \tag{3.11}$$

The following example illustrates how joint probabilities can be used to highlight correlation phenomena. This example features 200 devices, each with 64 CRPs, all exhibiting perfect Uniformity. However, challenges 10 and 42 exhibit positive correlation across all devices. Thanks to the individual analysis of the joint probabilities it's possible to have high granularity and know that they take the same values most of the time.

These joint probabilities may prove useful in the analysis of certain PUF desings, like RO-PUF, where a randomness source is compared multiple times. Abnormal values of the joint probabilities may be a good indicator of a problem with the randomness sources or the comparison method used.

Other proposals in the literature that are worth considered are shown [171] where the authors proposer the Correlation Sensitive Metric. A techniques similar to the matrix proposed here, albeit with different computations, is presented in [174]. In [175], the authors leverage correlation-spectra in Boolean functions and [176] where the authors propose a batter of tests to determine spatial correlations. Moreover, there are numerical techniques like LISA [177] that can help find the best physical arrangement of components to maximize the PUF's secret extraction.

3.2 Punctual Bitaliasing

This new metric integrates Bit-aliasing and Reliability into a unified measure, offering a comprehensive perspective on the behavior of the PUF in time. The concept revolves around predicting the Bit-aliasing of a subsequent sample given the PUF's Reliability and the Bit-aliasing of the initial enrollment. This prediction assumes no application of filtering or error correction techniques. Given that not all responses exhibit 100% Reliability, it is

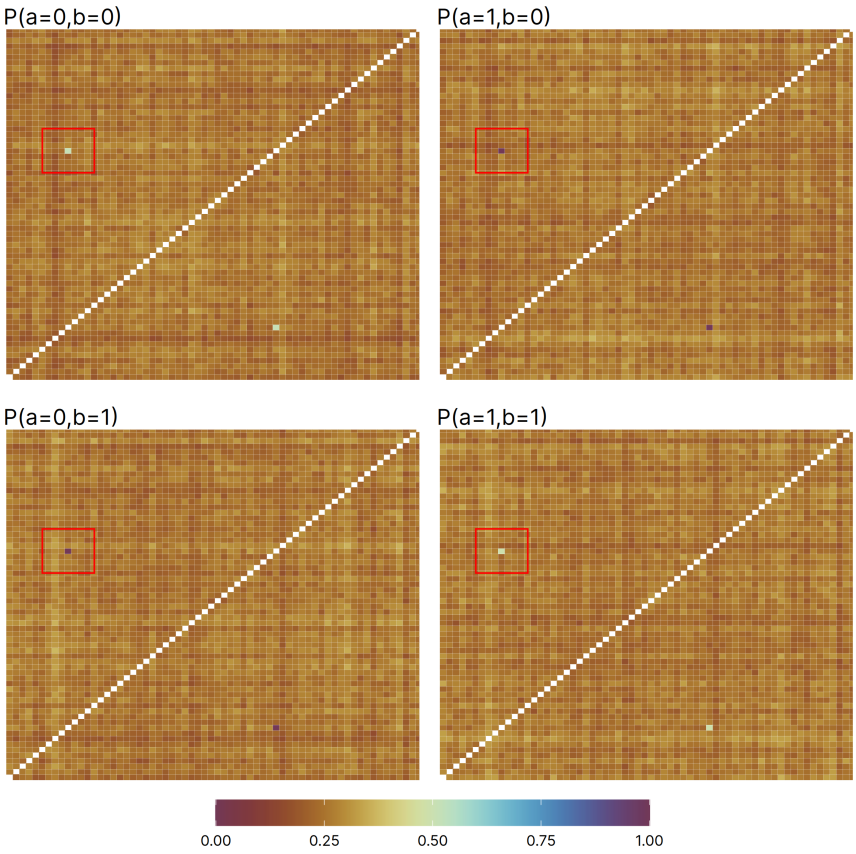


Figure 15: Example of Joint Probabilities

expected that the Bit-aliasing of the new sample will differ from that of the initial enrollment and this metric tries to quantify this effect.

It's computed as depicted in Equation (3.14). Analogous to the canonical Bit-aliasing its ideal value is 0.5. Through some algebra it shows that for this to happen $PBA = 0.5 \iff p = 0.5, p_{01} = 0, p_{11} = 1$. In fact, the probabilities p_{01} and p_{11} form the right column vector of the ideal Reliability Invariance matrix.

$$\begin{aligned} \text{Punctual Bitaliasing} = PBA &= P(1 \mid RI) && (3.12)\\ &= (P(1) \cap P_{11}) \cup (P(0) \cap P_{01}) && (3.13)\\ &= (p \times p_{11}) + ((1-p) \times p_{01}) && (3.14) \end{aligned}$$

The main limitation of this computation is that it does not take into account the probabilities p_{00} or p_{10}. This wouldn't suppose a problem if the probabilities were dependent on each other, but as said above, this does not need to be the case. Further work would include the inclusion of the rest of the probabilities into another metric computation.

The use of the Kullback-Leibler divergence is proposed to quantify the effect of Reliability on Bit-aliasing by analyzing the marginals of each response, represented by Punctual Bit-aliasing and canonical Bit-aliasing, as highlighted in Equation 3.15. Here, $\mathcal{A}$ represents the alphabet of the PUF, which for most PUFs corresponds to the possible binary values 0 and 1.

$$D_{KL}(PBA \parallel BA) = \sum_{a \in \mathcal{A}} PBA(a) \log\left(\frac{PBA(a)}{BA(a)}\right) \tag{3.15}$$

3.3 Entropy based metrics

We propose a series of metrics that use Shannon Entropy in order to mitigate the problems related to omitting the standard deviation in the representation of the metrics as proposed above.

Uniformity per device (UPD)

Analogous to Uniformity, UPD measures the the intra-device entropy by measuring the statistical distribution of all responses within a single device. We have defined UPD by applying the Shannon Entropy function, which provides the best score (i.e., 1) when the distribution is perfectly uniform (50% of 0s and 50% of 1s).

$$\text{UPD}(d) = \text{H}\left(\frac{1}{\#C} \sum_{r \in R_d} r\right) \tag{3.16}$$

Uniformity per challenge (UPC)

Analogous to Bit-aliasing, UPC evaluates the inter-device entropy by measuring the distributions of responses across devices for the same given challenge. As for UPD, the use of the Shannon entropy operator provides the best score (i.e., 1) when the distribution is perfectly uniform (50% of 0s and 50% of 1s).

$$\mathrm{UPC}(c) = \mathrm{H}\left(\frac{1}{\#D}\sum_{r\in R_c} r\right) \tag{3.17}$$

Reliable Entropy

We introduce a novel metric designed to integrate various metrics into a single value. This approach aims to address masking issues and provides valuable insights into suitable CRPs across all devices. This new metric, designate as Reliable Entropy, is defined in Equation 3.18.

$$RE = \min\left[\mathrm{H}(\text{Uniformity per bit}),\ \mathrm{H}(\text{Bit-aliasing})\right] - \mathrm{H}(\text{Reliability}) \tag{3.18}$$

The final goal of this metric is to identify the responses that have high Reliability while mantaining high Entropy, that is, ideal Uniformity or Bit-aliasing. To compute this metric, initially the Uniformity of each device is computed along with the Bit-aliasing of each response. Then, for each challenge, we define the *Uniformity per bit* as the device's Uniformity if that bit is 1 and the complementary value otherwise. That is to say, if a device has a Uniformity of 45%, a response of 1 will have a Uniformity per Bit of 45% and a response of 0 will have a Uniformity per bit of 55%. By selecting the minimum Shannon Entropy of the Uniformiy per bit and Bit-aliasing, we ensure that the CRP yields good properties across all devices. Lastly, the Shannon Entropy of the Reliability is substracted from the computed value. By doing this, we ensure to select the CRPs that have maximum Reliability.

By selecting the minimum Shannon Entropy of the Uniformity per bit and Bit-aliasing, we ensure that the CRP has good properties across all devices. Finally, the Shannon Entropy of Reliability is subtracted from this value, ensuring the selection of CRPs that maximize Reliability.

3.4 Test Suite for PUFs

Although many efforts in the literature aim to provide a standardized test methodology for PUFs [178], some researchers have resorted to using the NIST Test Suite [179]. The NIST Test Suite is a statistical test suite for random and pseudorandom number generators intended for cryptographic applications. While it was designed to evaluate random sources, PUFs were not considered in its development, so using it for PUF analysis is discouraged. NIST requires vast amounts of data from the same source to be effective. Collecting such

large datasets is a challenge for PUFs, so the results of the NIST suite may not accurately reflect the true behaviour of PUFs. This problem is highlighted in [180] where the authors argue that typically PUF responses are concatenated into a one-dimensional dataset in order to perform the NIST tests. The authors show that, by doing so, correlation phenonema are obfuscated. They prove this experimentally and show that the estimated min-entropy can differ by orders of magnitude. To solve, this, the authors propose a prediction scheme to accurately predict the min-entropy based on the worst-case scenario.

Moreover, statistical tests alone are not strong evidence of entropy. According to NIST, models should provide an explicit statement of the expected entropy and offer a technical argument for why the entropy rate can be mantained. No set of general-purpose statistical tests can measure the entropy per sample in an arbitrary sequence of values. A better approach is to understand the unpredictability of the noise source outputs, model it, and use the model to estimate the entropy. General-purpose tests on outputs should be run as a sanity check.

Therefore, we postulate that a test suite for PUFs is necessary. This test suite should be adapted to the different idiosyncrasies of various PUF families, as some sources of randomness may be biased or attacked in ways that other families are not. The metrics used to evaluate and compare PUFs can be a subset of the battery of tests required to fully assess a PUF's performance. For example, PUFs that generate bits over time should study time correlation, while PUFs that generate all bits simultaneously do not face this issue. Furthermore, this test suite should account for both static factors (e.g., devices, process variability) and dynamic factors (e.g., temperature, voltage, ageing).

Due to time constraints, it was infeasible to create a comprehensive test suite for PUFs. However, the methodologies proposed in this section and the rest of the document could serve as the foundation for developing a new test suite.

Conclusions

IV.4

In order to evaluate and assess the behaviour of a PUF design, a series of statistical tests or metrics are commonly used. The canonical metrics proposed by Maiti were described in detail. These foundational metrics are Uniformity, Bit-aliasing, Uniqueness, and Reliability. Uniformity evaluates the distribution of bits in the response of each device. Bit-aliasing studies the bias in the distribution of responses across different PUF instances. Uniqueness measures the ability of a PUF to produce unique responses for each device, while Reliability assesses the consistency of PUF responses under varying environmental conditions such as temperature, voltage and ageing. Following the description of these canonical metrics, a series of alternative proposals in the literature are studied and compared to the canonical ones.

Afterwards, the statistical limitations of the canonical metrics were analysed in detail. Several key issues were identified, such as the lack of a universally accepted standard or model for benchmarking PUF performance, making comparisons across studies challenging. Additionally, the absence of correlation and auto-correlation analysis means the canonical metrics do not consider the interdependence of bits within the PUF responses, potentially overlooking significant patterns, as clearly illustrated with examples. The metrics also neglect the theoretical and practical boundaries of PUF uniqueness. Furthermore, the insufficient reporting of standard deviation for metrics can lead to severely misleading conclusions about PUF performance.

To address these critical limitations, several mitigations and alternative metrics have been proposed. Among them, using deviation from ideal values instead of absolute metrics and new metrics, such as Stability, Reliability Invariance, and Punctual Bit-aliasing, aim to capture aspects overlooked by the canonical metrics. Furthermore, incorporating correlation and auto-correlation analysis can help identify and quantify dependencies within PUF responses, providing a more comprehensive evaluation framework.

Finally, the groundworks for a new comprehensive test suite tailored for PUFs, analogous to the NIST standards for cryptographic modules, were addressed. The proposed test suite aims to standardize PUF evaluation and ensure consistent and reliable benchmarking across the field. Although the development of this suite is beyond the scope of the current work due to time constraints, it is identified as a critical area for future research.

This foundational chapter presents a path toward more rigorous and standardized testing protocols that will enhance the reliability and comparability of PUF technologies.

V

On the Reliability of Differential PUFs

State of the art

V.1

The quality metrics described in Chapter IV assess the performance of a PUF across different dimensions, primarily related to the uniqueness of the generated secrets and the ability of the PUF to reproduce those secrets. While all metrics are crucial for understanding PUF behaviour, Bit-aliasing and Reliability are particularly significant for the adoption of PUFs in modern circuits. Due to common issues with current PUF implementations, only a limited number of devices can be reliably deployed in the field, and their costs often render them unsuitable for industrial applications, as shown by Pour et al. [181].

As a result, significant research efforts are currently focused on developing (i) techniques for reliability analysis and evaluation, and (ii) techniques for reliability improvement.

To the best of our knowledge, there are no significant works focusing on reducing Bit-aliasing effects in PUFs, as challenges that generate biased responses are usually filtered out. However, certain design techniques proposed by Valles Novo et al., Nguyen et al., and Garg et al. [182, 183, 184] optimize the design process to enhance PUF performance. These proposals suggest a potential interconnection between Entropy and Reliability.

1.1 Techniques for Reliability analysis and evaluation

Maes[1] in 2013 was among the first to demonstrate the trade-off between PUF reliability and its entropy. Real silicon data validate that some responses are more prone to unreliability than others, leading to the ability to study the full failure distribution of a PUF-based application. This concept is now widely accepted and has given rise to numerous reliability assessment methodologies.

Building on this notion, Schaub et al. [154, 185] in 2018 provide a generic probabilistic method for delay PUFs (RO-PUF, RO sum PUF, and Loop PUF), where the trade-off between Reliability and Entropy is modelized based on the signal-to-noise ratio (SNR), and validated by real measurements. They offer an analytical method to filter out unreliable responses, employing simplified models of delay distribution (due to fabrication-induced variability and thermal noise) to evaluate the SNR of a PUF response.

Martin et al. [186] in 2019 introduce a RO-PUF reliability evaluation metric based on FPGA-extracted data. In their study, the authors demonstrate the relationship between frequency differences and Reliability solely from experimental data, extracting the actual distribution of frequencies under fabrication-induced variability and evaluating the frequency fluctuation associated with variations in the operating environment (temperature

[1] R. Maes, 'An accurate probabilistic reliability model for silicon pufs,' in *International Conference on Cryptographic Hardware and Embedded Systems*, Springer, 2013, pp. 73–89.

and noise). Numerical and statistical models for studying the reliability of delay-based PUFs, especially RO-PUFs, can be found in [148, 97].

1.2 Techniques for Reliability improvement

Two main lines of work can be found in the literature: Filtering and masking techniques and Error Correcting Codes (ECC).

Filtering unreliable bits involves understanding PUF behaviour under various environmental conditions and ageing. It involves removing bits from PUF responses that have low Reliability. Bhargava et al. demonstrated the effectiveness of filtering in [187]. Additionally, Schaub et al. compared filtering to fuzzy extraction and found filtering to be more resource efficient for improving PUF reliability [188]. However, filtering requires extensive characterization to assess the reliability of each PUF bit under all possible operating conditions.

ECCs enhance reliability by using circuit redundancy to detect and correct unreliable PUF responses with helper data calculated during PUF enrollment [189, 190, 103]. While highly effective in ensuring reliability, ECCs are resource-intensive in terms of area and power consumption. Optimizing circuit layouts to minimize susceptibility to environmental variations and noise can also stabilize PUF behaviour, as demonstrated by Bhargava et al. and Saraza Canflanca et al. [191, 135]. Other promising design proposals include using convolution operations on SRAM-based PUFs [192].

However, when using ECCs, caution is necessary since the helper data might leak information about the PUF to potential attackers as already shown in [104]. Studies by Beguinot et al. and Danger et al. propose RO-PUF designs with reduced helper data to mitigate this risk [102, 173].

Another approach leverages specific physical phenomena, notably NBTI, is a common ageing effect in CMOS technology that can degrade PUFs over time, compromising their stability. However, research by Roelke et al. and Garg et al. suggests that inducing controlled ageing in transistors can improve PUF reliability [134, 184].

Collectively, these techniques significantly enhance the durability and dependability of PUFs, making them more suitable for long-term security applications.

V.2 Relationship between Frequency Difference and Reliability

Reliability is defined as the ability of the PUF to produce the same response for a given challenge under different operation conditions and ageing. Recall that in RO-PUF, the responses are generated by comparing two nominally identical ROs as already shown in Figure 2. By convention, if the frequency of the first RO is larger than the frequency of the second, the PUF response is 1, and 0 otherwise. If the two frequencies are very similar, the response is prone to be unreliable since a small shift in the frequency in one of the ROs due to noise or environmental conditions can alter the response. Therefore, an in-depth study of the distribution of all frequency differences can provide some insight about the Reliability of the PUF.

The comparison is usually done in the digital domain and since the signal from the RO is an analog signal, the frequency is usually measured by tallying up a counter, which counts at each rising edge of the RO, and the PUF response is obtained by comparing two counters. The sampling frequency needs to be fast enough to measure the fastest RO frequency in the RO-PUF. RO pairs whose frequency difference is close to 0 Hz require many samples in order to provide a measurable difference in the counter, that is until the frequency lag between the 2 frequencies becomes significant. However, frequency differences that are very large provide a meaningul difference in the counter early on.

Based on the general agreement, the oscillation frequencies of all ROs in the PUF can be fitted to a normal distribution $F \sim N(\mu, \sigma)$. Given 2 different frequency distributions $F_1 \sim N(\mu_1, \sigma_1)$ and $F_1 \sim N(\mu_2, \sigma_2)$, the distribution resulting from their difference will also be normally distribution according to Equation 2.1.

$$F_1 - F_2 \sim N\left(\mu_1 - \mu_2, \sqrt{\sigma_1^2 + \sigma_2^2}\right) \tag{2.1}$$

To refer to these frequency differences, we will use the notation Δ. Thus, F_Δ represents the distribution of RO frequency differences, while f_δ denotes an individual frequency difference.

In the case of the RO-PUF, the compared frequencies come from the same distribution, so the resulting frequency difference distribution can be summarised as shown in Equation 2.2.

$$F_\Delta \doteq N\left(0, \sigma\sqrt{2}\right) \tag{2.2}$$

Throughout this manuscript, in the analysis concerning Ring Oscillators, we will make extensive use of the "Coefficient of Variation" CV, which is defined as the ratio between

the standard deviation and the mean as shown in Equation 2.3.

$$CV_\Delta = \frac{\sigma_\Delta}{\mu_\Delta} \tag{2.3}$$

However, in most occasions it's reciprocal Z its preferred. While we use the notation of Z to refer to the reciprocal of the Coefficient of Variation, it's important to mention that it's not the z-score unless otherwise stated.

$$Z_\Delta = \frac{1}{CV} = \frac{\mu_\Delta}{\sigma_\Delta} \tag{2.4}$$

As suggested by[1], to discriminate the unreliable pairs, whose frequency difference is close to 0Hz, from the possible reliable pairs, a threshold T is defined such that pairs for which $-T < f_\Delta < T$ are considered unreliable, as displayed in figure Figure 16.

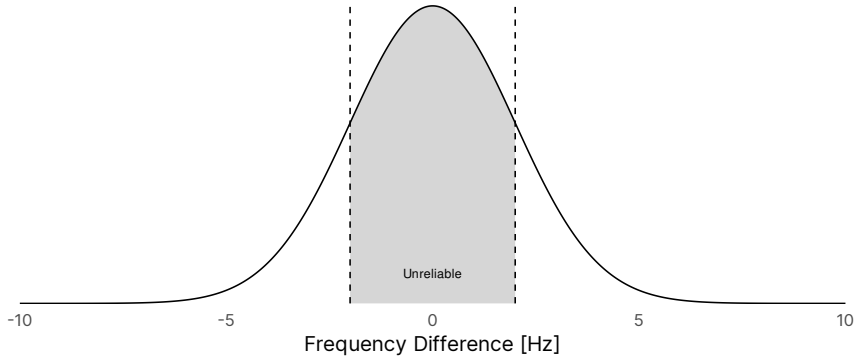


Figure 16: Relationship between Frequency Difference and Reliability

Given this threshold T, the Reliability of the PUF is then computed as the complement of the probability of unreliability as shown in Equation (2.8).

$$\begin{aligned}
\text{Reliability} &= \text{Total Area} - \text{Area Unreliability} && (2.5)\\
&= 1 - P_{Unrel} && (2.6)\\
&= 1 - \Phi\left(\frac{p-\mu_\Delta}{\sigma_\Delta}\right)\Big|_{-T}^{T} && (2.7)\\
&= 1 - \left[\Phi\left(\frac{T-\mu_\Delta}{\sigma_\Delta}\right) - \Phi\left(\frac{-T-\mu_\Delta}{\sigma_\Delta}\right)\right] && (2.8)
\end{aligned}$$

While this is generally agreed upon, the problem resides in finding the suitable threshold T to set the bounds of unreliability. For this, we propose a simulation-based framework

[1] H. Martin *et al.*, 'On the Reliability of the Ring Oscillator Physically Unclonable Functions,' in *2019 IEEE 4th International Verification and Security Workshop (IVSW)*, 2019, pp. 25–30. DOI: 10.1109/IVSW.2019.8854401.

which can be applied before manufacturing the PUF, which allows determining the relationship between frequency difference and Reliability. The proposed framework enables higher accuracy in the results since it is not based on predictive simplified models of the device variability and noise, but on actual technological electrical models. Moreover, this methodology offers some good oportunities as it is possible to perform online tests to assess the reliability of the response and can be used to implement filtering and masking techniques.

The methodology proposed is summarized in Algothim 1. This method does not work directly with the raw analog signals of the ROs but rather on the counter values computed to measure the frequency of the ROs. For every pair of ROs, at every counter time-step, we have calculated the response. We have done this for the nominal operation conditions (voltage V_n and temperature T_n) to determine the nominal response, as well as for different operation conditions to determine possible responses at run-time. If the response is equal to the one obtained at nominal conditions, it is considered reliable. The overall PUF reliability is calculated by averaging the reliability of all ROs. Thus, the reliability will be 100% if none of the responses differ from the nominal sample and 0% if all of them differ.

RO pairs for which the frequency difference is very small take more time to provide a meaningful counter difference. By saving the counter difference at every time sample we can provide valuable insights into the generated PUF responses. Indeed, RO evaluation time plays a huge role in the proper assessment of the Reliability of the PUF. In Chapter V.3, we describe a novel mathematical analysis to relate the frequency difference distribution of the RO to the approximate time needed to obtain a degree of reliability.

```
1  N = numTemps · numVDDs · (nRO/2)^2
2  for i ← 1 to (nRO / 2) do
3      for j ← 1 to (nRO / 2) do
4          for c ← 1 to nSamples do
5              # Calculation of nominal response
6              R_nom[i][j][c] ← sign(Diff(i, j, c, V_n, T_n))
7              # Calculation of reliability of each responses
8              foreach T in Temperatures do
9                  foreach V in Voltages do
10                     R ← sign(Diff(i, j, c, V, T))
11                     if R = R_nom then
12                         Rel[i][j][c] ← Rel[i][j][c] + 1
13                 end
14             end
15             # Calculation of PUF reliability
16             Rel_puf[c] ← Rel_puf[c] + Rel[i][j][c]/N
17         end
18     end
19 end
```

Algorithm 1: PUF Reliability calculation for threshold estimation.

To increase reliability, as demonstrated by[2], it is preferable to select pairs of ROs that overall widen the frequency difference distribution. However, challenges with a large frequency difference should not be used directly, since the bigger the frequency difference, the smaller the chance that process variation makes a difference in the different ICs. Those challenges will very like be common among multiple instances (i.e. they may present Bit-aliasing). Indeed, there is an inherent relationship between the Reliability and the Bit-aliasing (i.e. Entropy) of a PUF. This relationship will be studied in detail in Chapter V.1.

[2] M. T. Rahman *et al.*, 'A pair selection algorithm for robust ro-puf against environmental variations and aging,' in *2015 33rd IEEE International Conference on Computer Design (ICCD)*, IEEE, 2015, pp. 415–418.

Time To Response

V.3

There is a relationship between frequency difference and Reliability, concluding that frequency differences close to 0 Hz are deemed unreliable. While that is something agreed upon, the time needed to obtain reliable responses is rarely discussed. The analysis proposed here studies in detail the relationship between the oscillation frequency difference of a pair of ROs, the sampling frequency and the approximate time needed to obtain a response with a certain degree of confidence.

The analysis of the relationship between frequency difference and Reliability presented in the previous section can be expanded by taking into account the sampling frequency and the counter differences to provide an overall evaluation of the Reliability of the RO-PUF can be obtained. While this type of analysis is often omitted in the literature, it could yield critical information valuable for design choices for time-constrained environments. In addition, this analysis could be adapted to the intricacies of other delay-based PUFs by following the methodology proposed here.

For this analysis, we make the following assumption during the interrogation process, where are a pair of ROs is compared to yield a response. Both ROs of the pair being interrogated start oscillating at the same instant. Any delay in any of the RO due to delay in the lines or resistive defects is negligible. To guarantee that we allow for delays, and start counting when both ROs are oscillating (i.e. the counter enable is delayed relative to the RO enable). There are also effects that can alter the amplitude of the signal, but we are only interested in the frequency. Furthermore, if the frequencies are sampled in a period long enough, random effects like RTN are averaged out and cancelled out, so these type of random effects are not considered.

We consider the case of 2 ROs with different oscillating frequency, and compute the resulting signal from substracting both signals as shown in Figure 17. If both signals start oscillating at the same time, the difference signal will experience simultaneous zero-crossing periodically at intervals T_{sync}. On a mathematical level, the resulting frequency difference signal is the result of the difference of two "travelling waves" and the exact behaviour of the signal is given in Equation (3.2).

$$y(x,t) = y_m \sin(k_1 x - \omega_1 t) + y_m \sin(k_2 x - \omega_2 t) \tag{3.1}$$

$$= 2y_m \cos\left[\frac{k_1 - k_2}{2}x - \frac{\omega_1 - \omega_2}{2}t\right] \sin\left[\frac{k_1 + k_2}{2}x - \frac{\omega_1 + \omega_2}{2}t\right] \tag{3.2}$$

The resulting signal from the difference of the RO frequencies has multiple frequency components, but we are only interested in the one that controls the repetition interval. The interval T_{sync} at which both signals experience the zero-crossing is given by the period

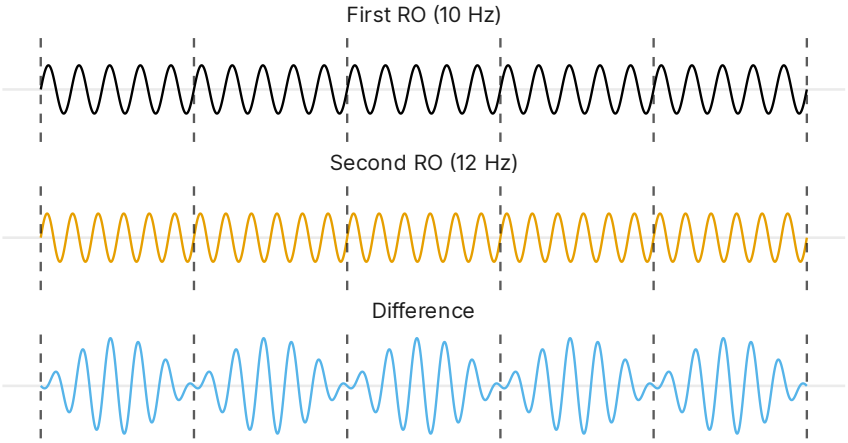


Figure 17: RO signals and the resulting Beat from their difference

of the frequency difference $f_\Delta = |f_1 - f_2|$. In the example shown in Figure 17 where both signals oscillate at 10 Hz and 12 Hz respectively, the interval T_{sync} is computed as follows:

$$T_{sync} = \frac{1}{12\,\text{Hz} - 10\,\text{Hz}} = \frac{1}{2\,\text{Hz}} = 0.5\,\text{s}$$

This implies that we expect changes in the counter difference every 0.5 seconds for this specific pair. With a sampling interval of 0.1 seconds, we would observe 5 consecutive identical values of the counter difference before detecting a change, such as 1, 1, 1, 1, 1, 2, 2, 2, 2, 2. Based on this observation, we can estimate the necessary number of samples to achieve a "reliable" measurement. Assuming a fixed sampling frequency t_{sample}, we define the *expected number of samples until the counter difference changes* as detailed in Equation 3.3. Expanding on this analysis allows us to establish a relationship between the anticipated response time and the reliability of the corresponding response, determined by the threshold computed using the methodology proposed in Chapter V.2.

$$E_{samples} = \frac{T_{sync}}{t_{sample}} = \frac{1}{f_\Delta \cdot t_{sample}} \tag{3.3}$$

This computation can then be extended to every pair of ROs in the PUF to analytically obtain the distribution of time expected to gather the responses, as illustrated in Figure 18. In the proposed example three distributions are shown, A, B and C, each one with wider distribution of frequencies. The top subfigure represents the absolute value of the frequency differences given by the 3 distributions. The bottom figure, represents the distribution of T_{sync} intervals computed on every pair. We can see that in distribution A, being the narrowest distribution, most frequency differences are very close to 0, so the repeating interval is very large. On the contrary, in distribution C, there are frequency differences that are much larger and consequently have a much lower T_{sync}.

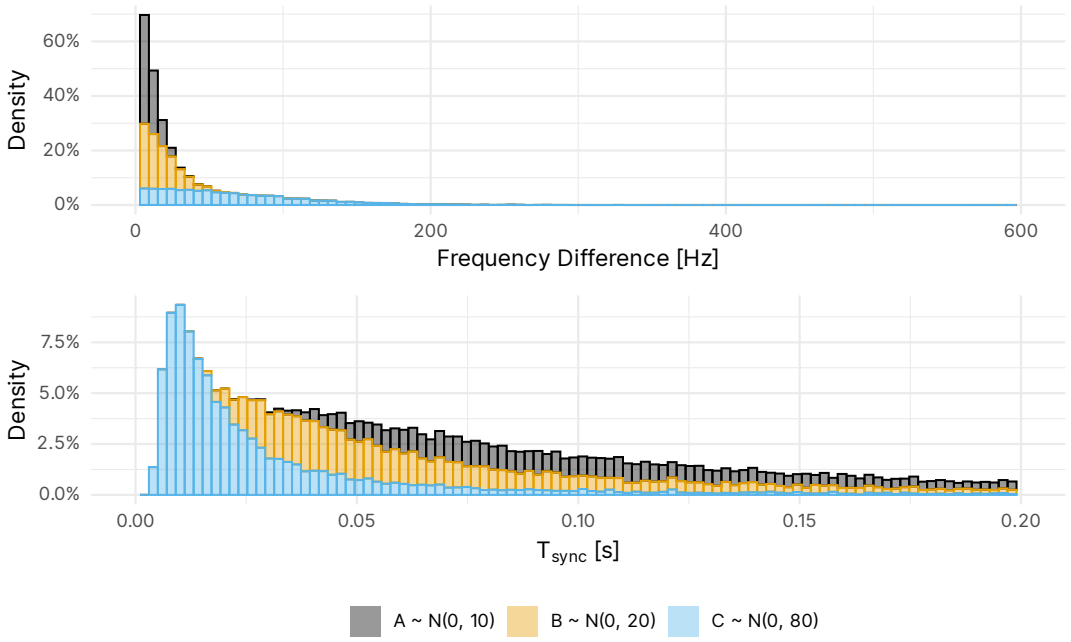


Figure 18: Frequency difference distributions and their corresponding distribution of repating intervals

3.1 Numerical estimation of the number of samples

We have shown that for a given RO pair being sampled at a fixed rate, the expected number of samples needed to see changes in the counter difference can be computed with the repeating interval T_{sync} of the RO pair. Even though this can be done numerically given all the frequency differences in the RO-PUF, from a design point of view, it would be beneficial to know how the shape of both distributions (i.e. RO frequencies and expected time) relate to each other. This analysis has been carried out numerically through a grid search implemented in Julia, whose code is provided in E.3.

The frequency difference F_Δ follows a normal distribution related to the frequency distribution of the ROs.

Since we are interested in measuring time, we can use the symmetric property of the normal distribution and create a folded normal distribution by taking the absolute frequency difference $Y \sim |F_\Delta|$ which can be modelled through a Beta distribution as shown in Equation 3.4.

$$|F_\Delta| \sim Gamma(\alpha_\Delta, \beta_\Delta) \tag{3.4}$$

In the case where $\mu = 0$, then we can simplify the study by using a half-normal distribution, which is a special case of the generalized Gamma distribution.

$$|F_\Delta| \dot{\sim} IG(\mu, \delta) \iff \mu = 0 \tag{3.5}$$

The distribution of T_{sync} intervals for all frequency differences defined by F_Δ will be denote as TTR, for "Time To Response". As presented above, the time to response is inversely proportional to the frequency difference.

$$TTR \sim \frac{1}{|F_\Delta|}$$

The TTR distribution, being the inverse of a folded normal distribution, can be modelled by using a Wald distribution (inverse Gaussian distribution), but we can also fit a gamma, which should be simpler to work with. There are some instabilities and expensive numerical computations of the Wald distributions that make the Gamma distribution much easier to work with, so we suggest using the Gamma distribution instead.

$$TTR \dot{\sim} Gamma(\alpha_{TTR}, \beta_{TTR})$$

Given the distributions $|F_\Delta|$ and TTR, our goal is to establish the connection between the shape of the F_Δ distribution given by $Z_\Delta = \mu/\sigma$ and the parameters $\alpha_\Delta, \beta_\Delta$ and $\alpha_{TTR}, \beta_{TTR}$ respectively. For analytical simplicity, we will make use of the $Gamma(k, \theta)$ parametrization, which corresponds to $Gamma\,(\alpha, 1/\beta)$.

An initial exploration of this relationship is depicted in Figure 19, where the left column illustrates the relationship between Z_Δ and $|F_\Delta|$, while the right column shows the relationship between Z_Δ and TTR. It is observed that both k_Δ and k_{TTR} exhibit a parabolic shape with a slight dependency on σσ. Conversely, θ_{TTR} also displays a parabolic shape, yet its dependence on σ is more pronounced. This outcome is expected as demonstrated in Figure 18, where narrower frequency difference distributions correspond to wider distributions of T_{sync}, reflected by larger θ.

Furthermore, k_Δ demonstrates two distinct behaviours: a linear decrease for very low values of Z_Δ, and an increase for larger values, where the slope is influenced by σ.

Given the relationship shown, we fit a quadratic model for α_Δ and 2 linear models for k_Δ. In the case of the TTR, two quadratic models are fitted. The following models are obtained for the distribution of absolute frequency differences.

$$k_\Delta = |Z_\Delta^2 - 1| \qquad R^2 = 0.9955 \tag{3.6}$$

$$\theta_\Delta = \begin{cases} \frac{|\mu|}{\sigma^2} & \text{if } |Z_\Delta| \geq 1 \\ \frac{1.75}{\sigma} - \frac{0.45|\mu|}{\sigma^2} & \text{otherwise} \end{cases} \qquad R^2 = 0.9975 \tag{3.7}$$

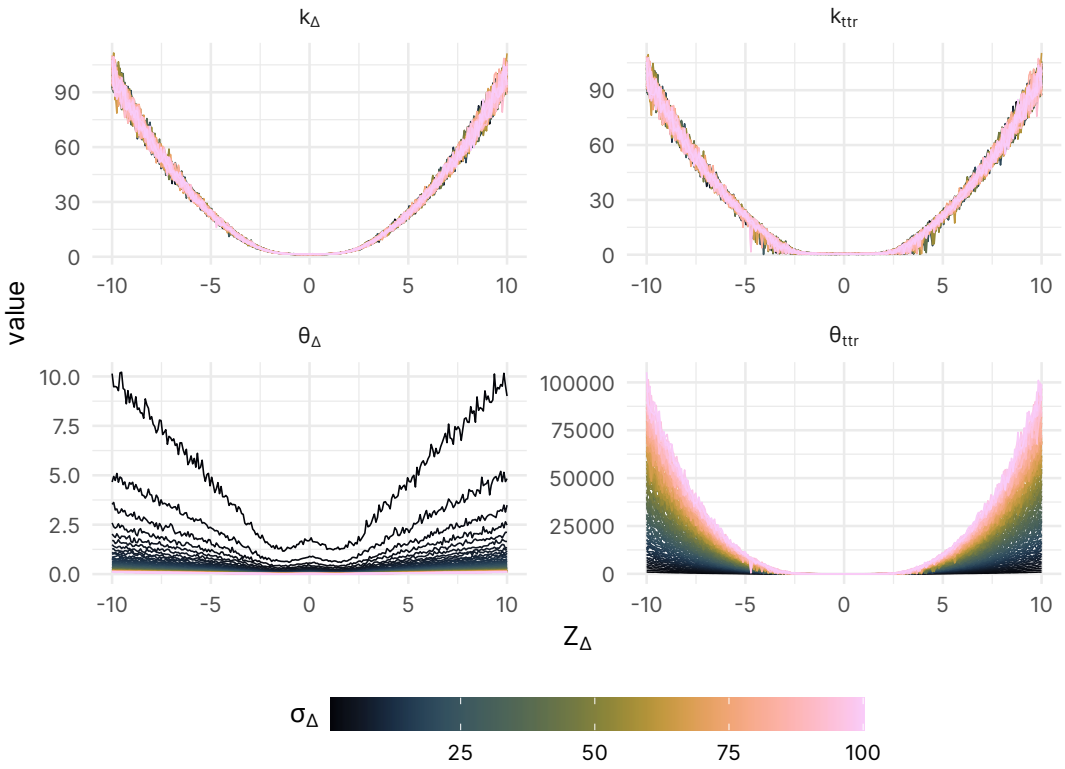


Figure 19: Relationship between Z_Δ and the different parameters

The analysis of the parameters of the TTR distribution yields the following models

$$k_{TTR} = Z_{\Delta}^2 - 3.12 \qquad R^2 = 0.9954 \tag{3.8}$$

$$\theta_{TTR} = 18.6 \cdot \frac{\mu^2}{\sigma} - 520 \cdot \sigma \qquad R^2 = 0.9724 \tag{3.9}$$

3.2 Conclusions

The methodology proposed here allows for the study of the relationship between the frequency difference distribution obtained in an RO-PUF and the distribution of time needed to obtain reliable responses. This analytical models should work in tandem with the methodology proposed in the previous section to improve or at least study the Reliability of an RO-PUF. Once the bounds of unreliability have been computed for a certain RO-PUF, we can estimate the time that corresponds to the computed bounds.

It's important to state that these are not mathematical derivation of the real behaviour, but approximations. These models present heteroscedasticity for very large values of Z_{Δ}. A proper mathematical analysis will be needed to order to derive the real expressions of the distributions in terms of the distribution of frequency differences.

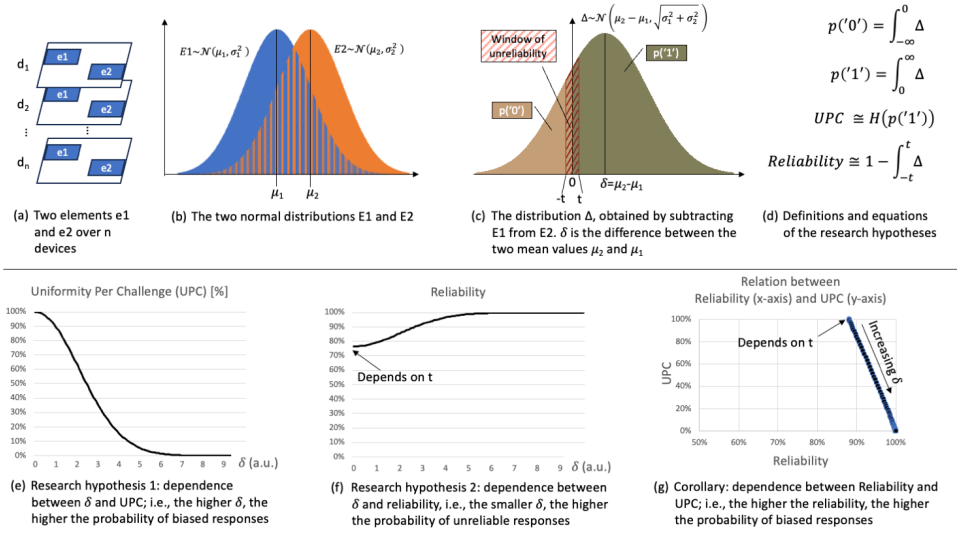
VI

On the Relationship between Reliability and Entropy of Differential PUFs

VI.1 Relationship between Reliability and Entropy

In Chapter VI.2 we have proposed that the Reliability of a PUF is proportional to the frequency difference. In this section, we delve into the relationship between Reliability and Entropy. For this, we focus on PUFs whose responses are generated by comparing nominally-identical physical quantities, which in our case is the RO-PUF.

We consider the case where 2 ROs have measurement values distributed among all devices, as normal distributions: E_1 and E_2. Therefore their difference is distributed following a Normal distribution as already shown in Equation 2.1 and depicted in Figure 20. In the case only random variability is present, the average value of the two frequency distributions should have the same value thus leading to an equal probability of responses at 0 and 1. Nevertheless, because of systematic process variability and design choices, the two values might differ, in which case the probability of one of the responses becomes predominant.



(a) Two elements e1 and e2 over n devices

(b) The two normal distributions E1 and E2

(c) The distribution Δ, obtained by subtracting E1 from E2. δ is the difference between the two mean values μ_2 and μ_1

(d) Definitions and equations of the research hypotheses

(e) Research hypothesis 1: dependence between δ and UPC; i.e., the higher δ, the higher the probability of biased responses

(f) Research hypothesis 2: dependence between δ and reliability, i.e., the smaller δ, the higher the probability of unreliable responses

(g) Corollary: dependence between Reliability and UPC; i.e., the higher the reliability, the higher the probability of biased responses

Figure 20: Description of the relationship between frequency difference and Entropy on Differential PUFs.

This bias in the probability of the response can be studied by analysing the distance of the frequency distribution from 0 as shown in Figure 21. If the difference of the mean values of the frequency differences is positive, the probability of 1 is larger than the probability of 0. On the contrary, if the difference is negative, the probability of obtaining a 0 is larger than the probability of obtaining a 1. This uneven ratino of probabilities leads to a decrease of Entropy, degrading the performance of the PUF. In Figure 20 the bias is depicted by means of the Uniformity Per Challenge (UPC) metrics described in Chapter VI.2.

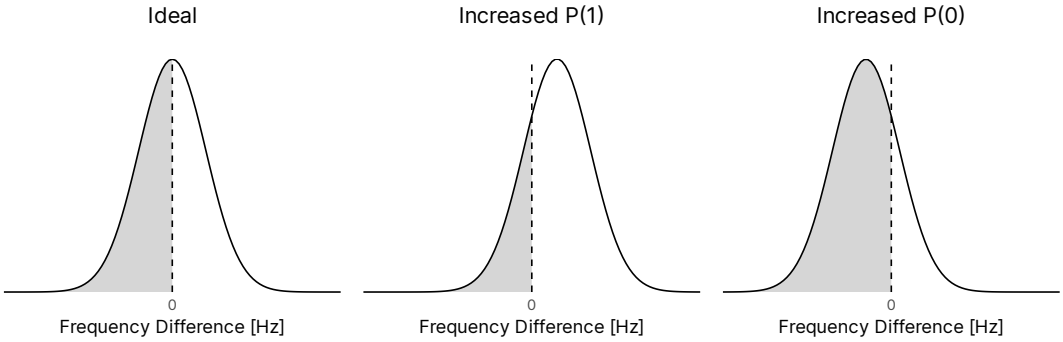


Figure 21: Relationship between the frequency difference distribution and Bit-aliasing

Since both the Reliability and Bit-aliasing are related to the frequency difference, the direct consequence is that both metrics of the PUF are inverse proportional so that large Reliability implies low inter-device Entropy and vice-versa. Indeed, for differences of the mean frequency of the two ROs different from 0, the window of unreliability slides towards the extremes of the F_Δ distribution, therefore the area of concern becomes smaller, increasing thus the reliability of the PUF. Consequently, we now not only have the area of unreliability at the centre of the distribution as depicted in Figure 16, but also the area of bias at the extremes of the distribution, as shown in Figure 22.

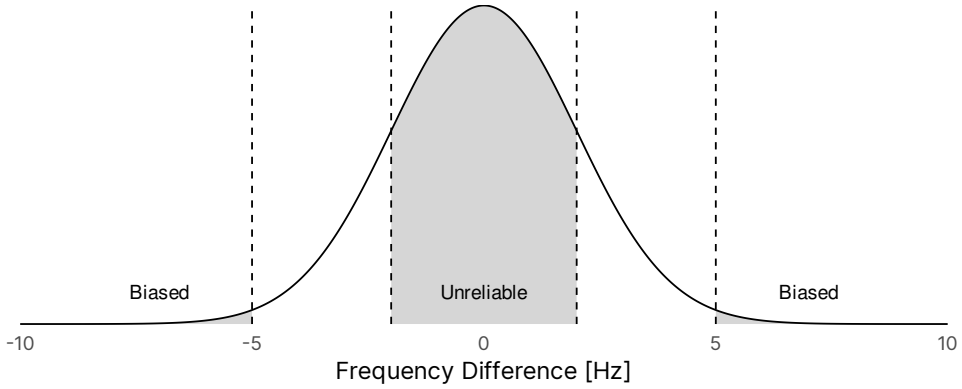


Figure 22: Relationship between Frequency Difference, Reliability and Entropy

This relationship has been proven by the simulation method proposed in Chapter VI.2. The results from the Infineon dataset strongly supports this analysis as it will be shown in detail in Chapter VI.2, Chapter VI.3 and published in[1].

[1]V. Kulagin *et al.*, 'On the Relation Between Reliability and Entropy in Physical Unclonable Functions,' *IEEE Design & Test*, pp. 1–1, 2024. DOI: 10.1109/MDAT.2024.3425791.

VI.2 Simulation results and mitigation techniques

In this section, we conduct an in-depth analysis of the simulation data to evaluate the filtering technique described in Chapter VI.2. The analysis and results discussed here have been previously published in [6] and subsequently extended in [7, 8].

Recall that for each simulated RO, we record the supply voltage, temperature, and counter value at specific timesteps. For each RO pair, we compute the counter difference at each timestep and calculate the reliability of that pair accordingly. This allows us to correlate the counter difference with the reliability for each pair at every timestep. Figure 23 illustrates the counter differences achieved for all pairs and the reliability of each RO pair at a particular timestep. It is evident from this Figure that, at this specific timestep, RO pairs with a frequency difference greater than 2 achieve a reliability of 100%.

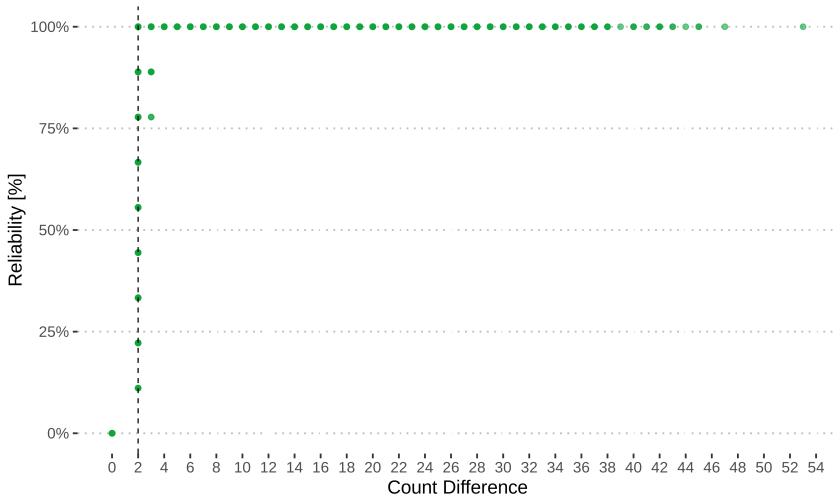


Figure 23: Reliability of the RO-PUF at a single timestep

Subsequently, for each timestep, we determine the minimum counter difference that achieves maximum reliability (e.g., 2 in the previous example) for all RO pairs across all timesteps. In cases where different RO pairs exhibit diverse reliability values for the same counter difference, we select the minimum counter difference that ensures all RO pairs achieve maximum reliability (e.g., 4 in the previous example). These thresholds at every timestep are then employed to filter the responses based on the measurement time.

Figure 24 showcases these results for our simulations. Recall that the RO is controlled via an enable signal with a NAND gate. Once the signal is enabled, the RO requires approximately 30 ns to reach transient mode, resulting in no data from 0 to 30 ns. The purple dots represent

the maximum threshold for all pairs at each timestep, while the green line represents the fitted line used as the threshold. If an RO pair provides a counter difference above this line, the pair is considered reliable; otherwise, it is deemed unreliable. Additionally and to ensure the reliability of the response, two separate measurements can be performed at different timesteps, with both needing to exceed the threshold.

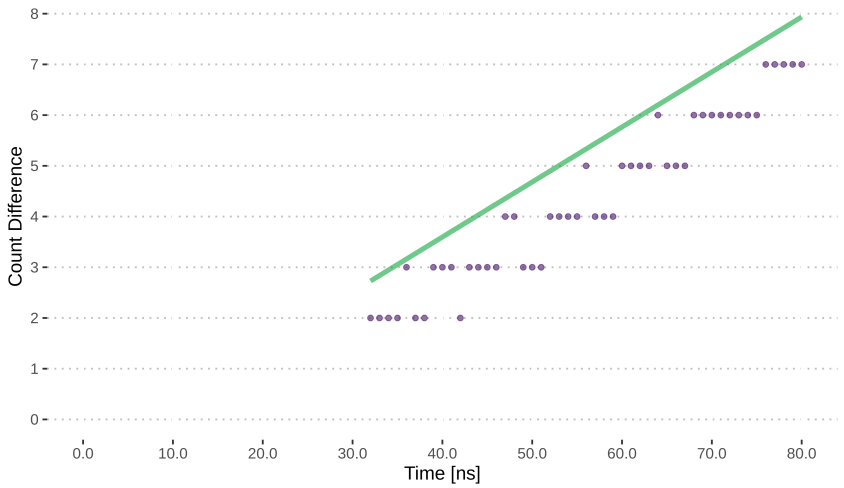


Figure 24: Reliability of the RO-PUF at different time steps

Now we select a specific time sample for detailed analysis and apply the proposed filtering technique. By varying the counter difference threshold, from 0 to 10 in our case, we monitor the number of RO pairs whose counter difference exceeds the selected threshold. The reliability of the RO-PUF is recalculated after filtering out the masked pairs. A threshold of 0 implies no masking. The results are presented in Figure 25. It is observed that increasing the counter difference threshold reduces the number of available responses by filtering out those closer to the boundaries of unreliability. However, this enhances the overall reliability of the PUF as only reliable challenges are selected. In this case, a threshold of 4 achieves a global reliability of 100% while retaining 76% of the RO pairs.

The methodology is further extended to incorporate Entropy. Using the same approach, we compute the Reliability and Entropy of the RO-PUF at each timestep. The results shown in Figure 25 are now supplemented with these new findings and depicted in Figure 26. Instead of the number of valid CRPs, we display the number of invalid CRPs. As demonstrated, increasing the counter difference threshold enhances Reliability. However, the Entropy of the PUF decreases with higher thresholds. Larger thresholds mask unreliable responses, but retain responses with very large frequency differences, resulting in reliable but potentially bit-aliased responses. For thresholds of 3 or 4, the studied PUF design achieves a favorable balance between reliability and entropy, with only a minor reduction in the number of CRPs.

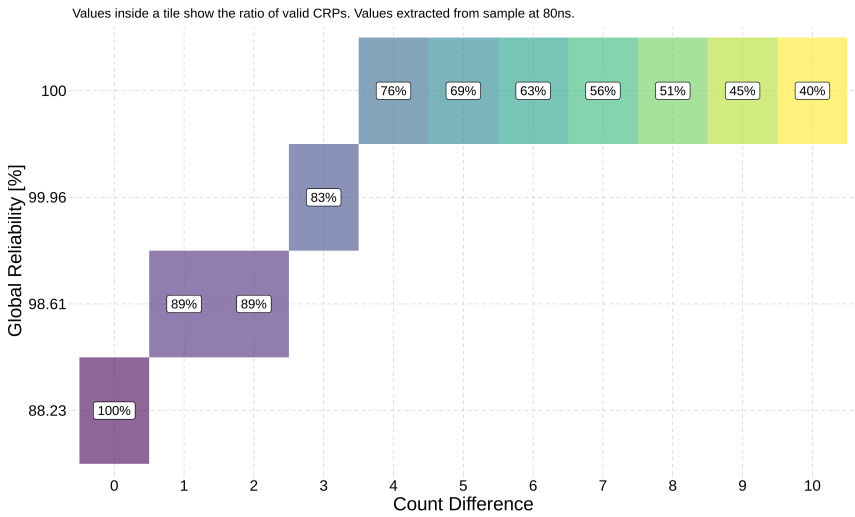


Figure 25: Relationship between RO-PUF Reliability, counter difference threshold and valid number of CRPs

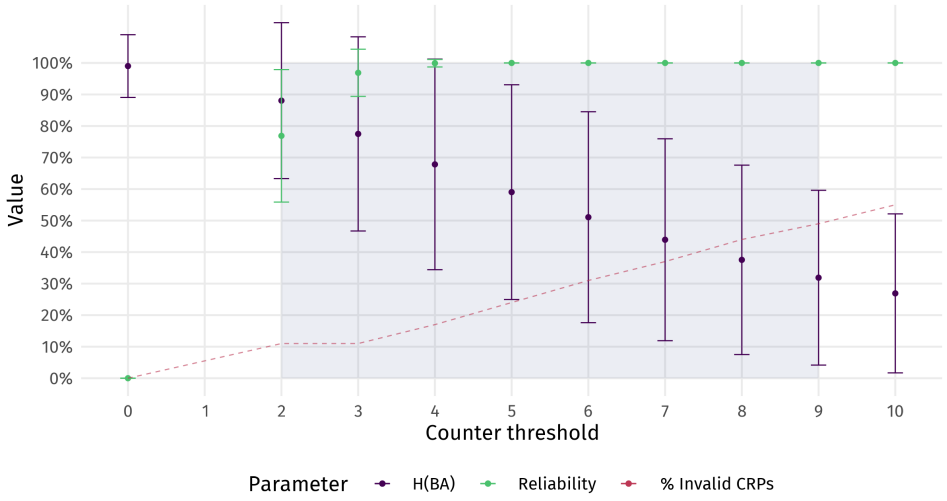


Figure 26: Relationship between RO-PUF Reliability, Entropy, counter difference threshold and valid number of CRPs

Analysis of experimental data

VI.3

In this section we provide an in-depth analysis of the SRAM data gathered with the SRAMPlatform as well as the dataset of the industrial circuit provided by Infineon.

Although PUF keys typically range from 128 to 256 bits in length, each memory studied contains approximately 80 kilobytes (80 kB) of data, totaling around 655360 bits. This provides extensive opportunity for thorough analysis, although some analyses and figures can become quite complex.

3.1 SRAM analysis

3.1.1 Metadata analysis

We start first with an analysis of the metadata of the SRAMs. Recall that STM32 devices contain a unique UID provided at manufacture. This manufacture contains information about the wafer number, lot number and physical coordinates of the chip in the wafer. This information is shown in Figure 27. We don't have access to information about the wafer so we don't know exactly the correct size. We assume here that all coordinates are positive, (negative numbers cannot be encoded in the UID) and the original wafer size is drawn in the circle. However, we have no way to confirm this so this remains a hypothesis.

Secondly, Figure 28 displays the different values of supply voltage and temperature obtained from the measurements. It is evident that all devices exhibit a consistent trend. However, there is a notable period (28-11-2022) where the values become erratic. While the exact cause is unknown, we consider two plausible explanations:

Firstly, it could be due to sensor calibration issues, as these devices are intended for prototyping and the sensors may not be entirely reliable.

Secondly, external factors may be at play. The station is located in a room with other stations and devices, adjacent to a window, potentially subjecting it to environmental influences that could affect certain measurements.

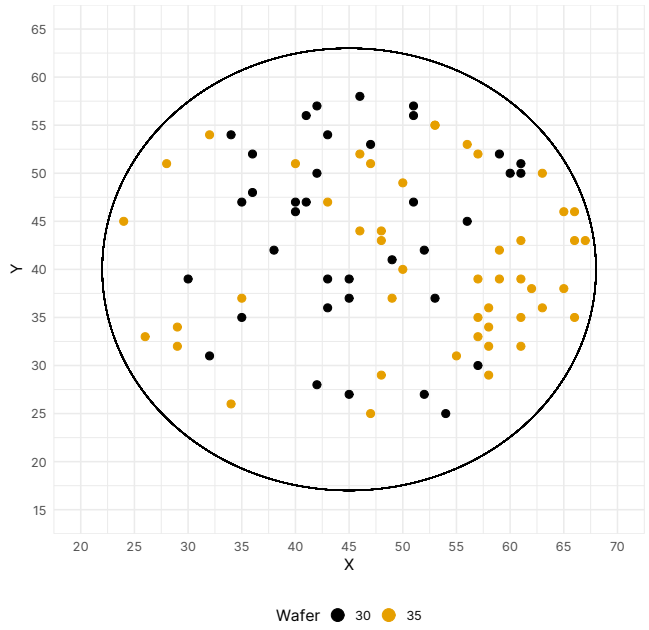


Figure 27: Wafer positions of the STM32 devices

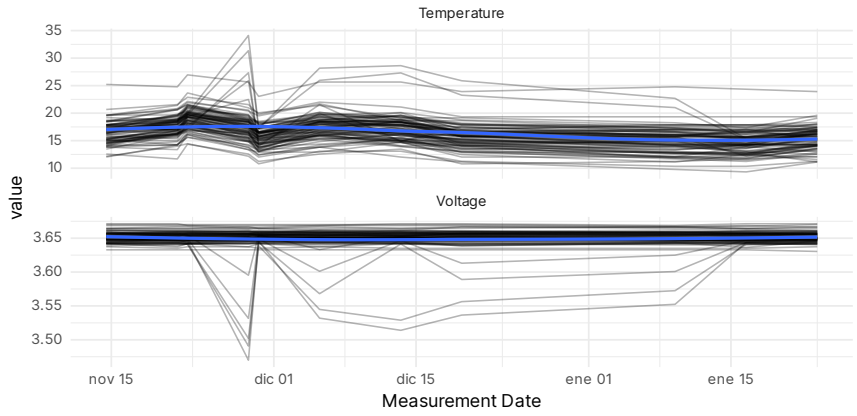
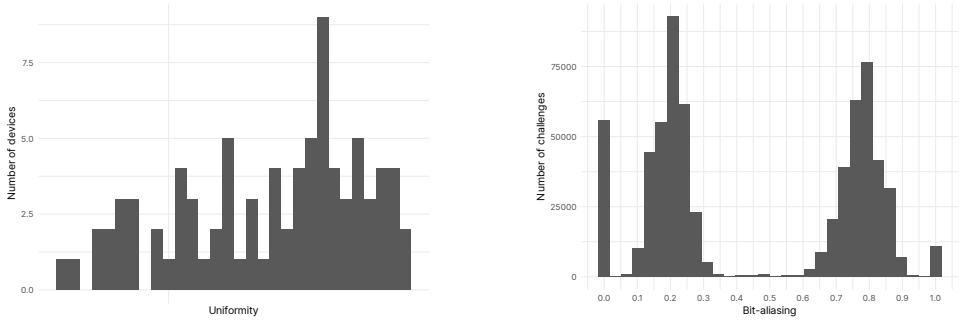


Figure 28: Temperature and voltage information of the STM32 devices. The blue line represents the trend behaviour.

3.1.2 Canonical metrics evaluation

The Confidence Interval of Uniqueness has been computed through bootstrap. In this case we can also compute them through the t-test.



(a) Uniformity histogram

(b) Bit-aliasing histogram

(c) Entropy histogram

(d) Uniqueness histogram

Figure 29: Histogram of SRAM-PUF metrics

We study now in detail the distribution of Bit-aliasing due to the extreme range and deviation obtained. Figure 30 presents a scatter plot where the X axis represents the canonical Bit-aliasing and the Y axis represents the logical address.

We observe immediately the first problem. The values of Bit-aliasing are gathered around 0.25 and 0.75. When averaging out the results, we obtain a mean Bit-aliasing of 0.5, but

Table 14: Summary of the SRAM-PUF metrics

Metric	μ	σ	$CI_{95\%}$	[min, max]
Uniformity	0.4556	0.0052	[0.4540, 0.4571]	[0.4443, 0.4628]
Bit-aliasing	0.4556	0.3268	[0.4548, 0.4564]	[0, 1]
Uniqueness	0.2893	0.0612	[0.2852, 0.2934]	[0.0219, 0.3858]

these spatial phenomena are masked if the standard deviation is not provided, as it was shown in Chapter VI.2.

Moreover, we can observe the following additional behaviour. The rectangles situated at the extremes marked in red are filled mostly with 0s since it contains program data and the memory stack. This is something unavoidable as the boards ultimately need to run some software. We still have a lot of usable area left in the boards. These areas will be ignored from here on since it they won't provide any useful information.

At address near 48 kiB (logical address 0xC000) there seems to be a "spike" of Bit-aliasing. This effect will be apparent in the future. Since we don't have any information about the physical layout of the SRAM, we suppose it's due to

We now rearrange the data into a new format to see the apparent result. The new rearrangement groups bits in blocks of 512 bits, resulting in a total of 1280 groups, as shown in Figure 31. We have to state that we don't know the physical address of the data, only the logical so this representation used is completely arbitrary.

Now the aliasing pattern is extremely clear. There is an alternating pattern of high and low Bit-aliasing repeating every 32 bits. This is extremely intriguing as these devices have an architecture of 32 bits. Moreover, the fist 32 bits out of each 256 blocks seems to have an average Bit-aliasing a bit smaller than the other blocks of 1s.

We can also see a red line that crosses horizontally at around block 768 which corresponds to the logical address 0xC000 or 48 kiB, which is the same "spike" as in Figure 30.

The Auto-correlation matrix proposed in part IV has been computed for the SRAM and is shown in Figure 32. Due to the spatial repetition of results in the SRAM, the matrix has been computed per device in blocks of 512 bits and the results have been averaged out across all devices.

The checkboard pattern is indicative of the weird repetition pattern that has been observed before. However, barely distinguishable, there are a group of bits in the first 32 bits of each block that have better auto-correlation. This is indicative that these few bits may pose good PUF qualities. This also showcases the very poor yield of the SRAM, where only a handful of bits in a block of 512 bits do not present correlation phenomena.

The same pattern can be seen also by studying the Kullback-Leibler Divergence on each challenge as shown in Figure 33, considering that the reference distribution is the ideal binomial with Bit-aliasing of 0.5. However, this new study seems to indicate that the pattern is repeated every 256 bits instead of 512. However, since the bits with good quality are in the first 32 bits of the 512 bits block, we stick with blocks 512 bits wide.

The joint probabilities derived from the AC matrix have also been computed and showcased in Figure 34. The fine granularity of this study shows there is a "preferred" correlation between responses, specially in the bits in the range of 96 to 192 and 416 to 480. Someone with expertise on the SRAM design field could probably extract information about the SRAM layout from this analysis.

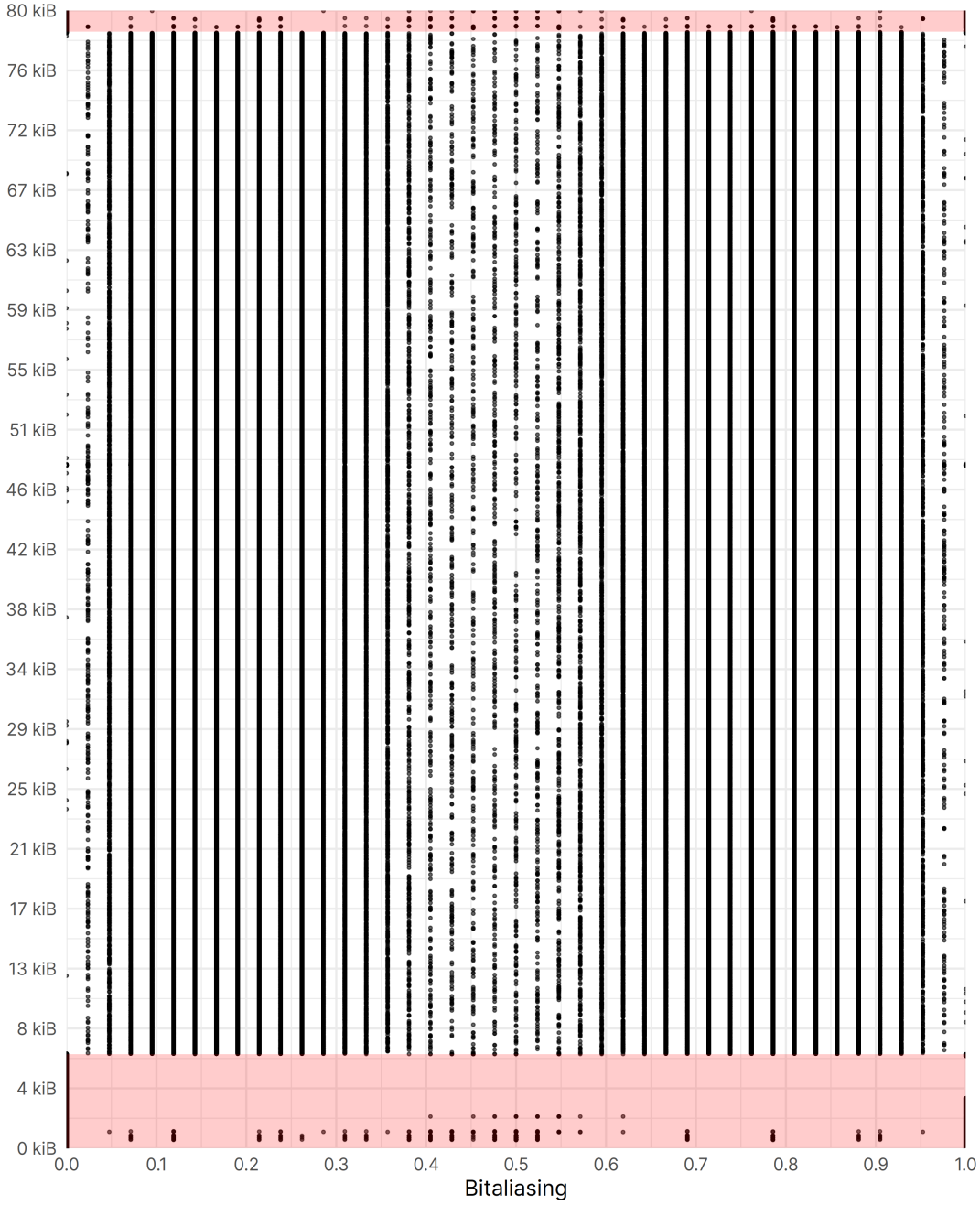


Figure 30: Scatter plot of Bit-aliasing across all bits in the SRAM

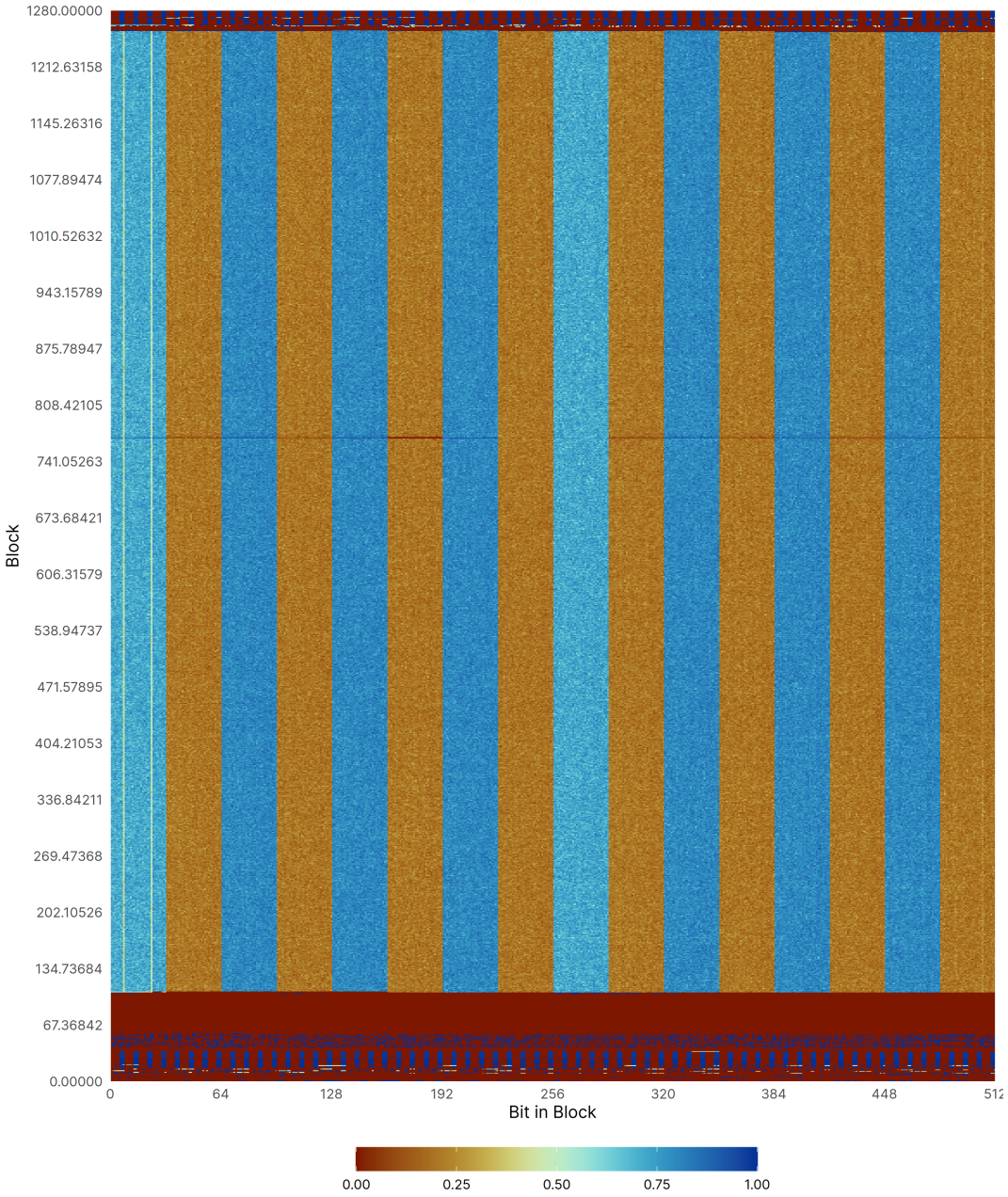


Figure 31: Heatmap of Bit-aliasing across all bits in the SRAM

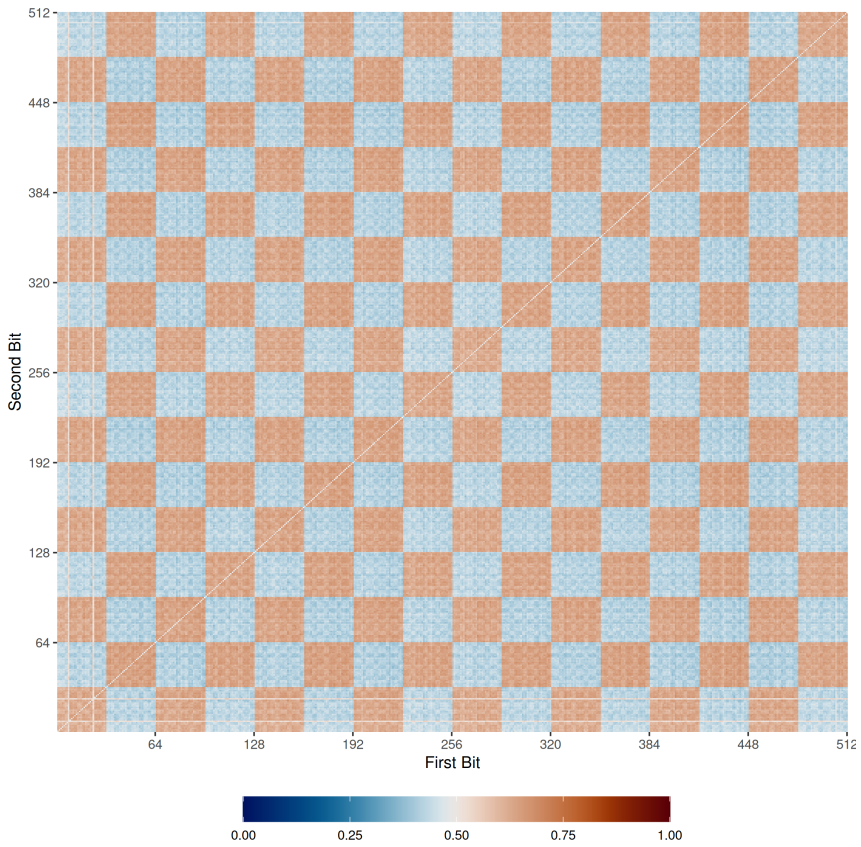


Figure 32: Average Auto-Correlation matrix of the SRAM. The matrix has been computed per block of 512 bits and averaged out.

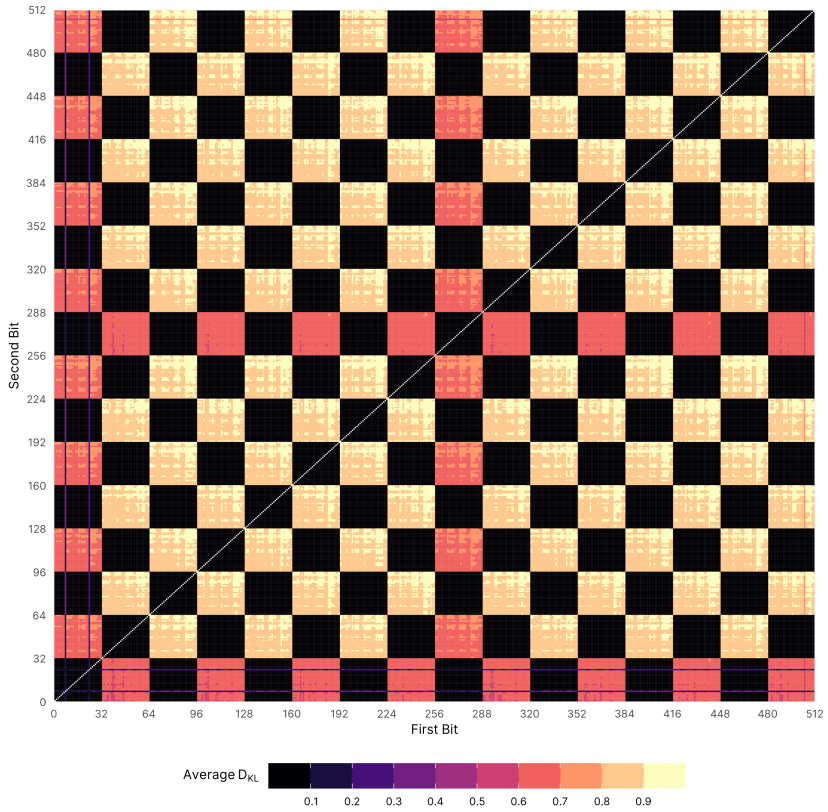


Figure 33: Average Kullback-Leibler divergence of the SRAM. The divergency has been computed per blocks of 512 bits and averaged out.

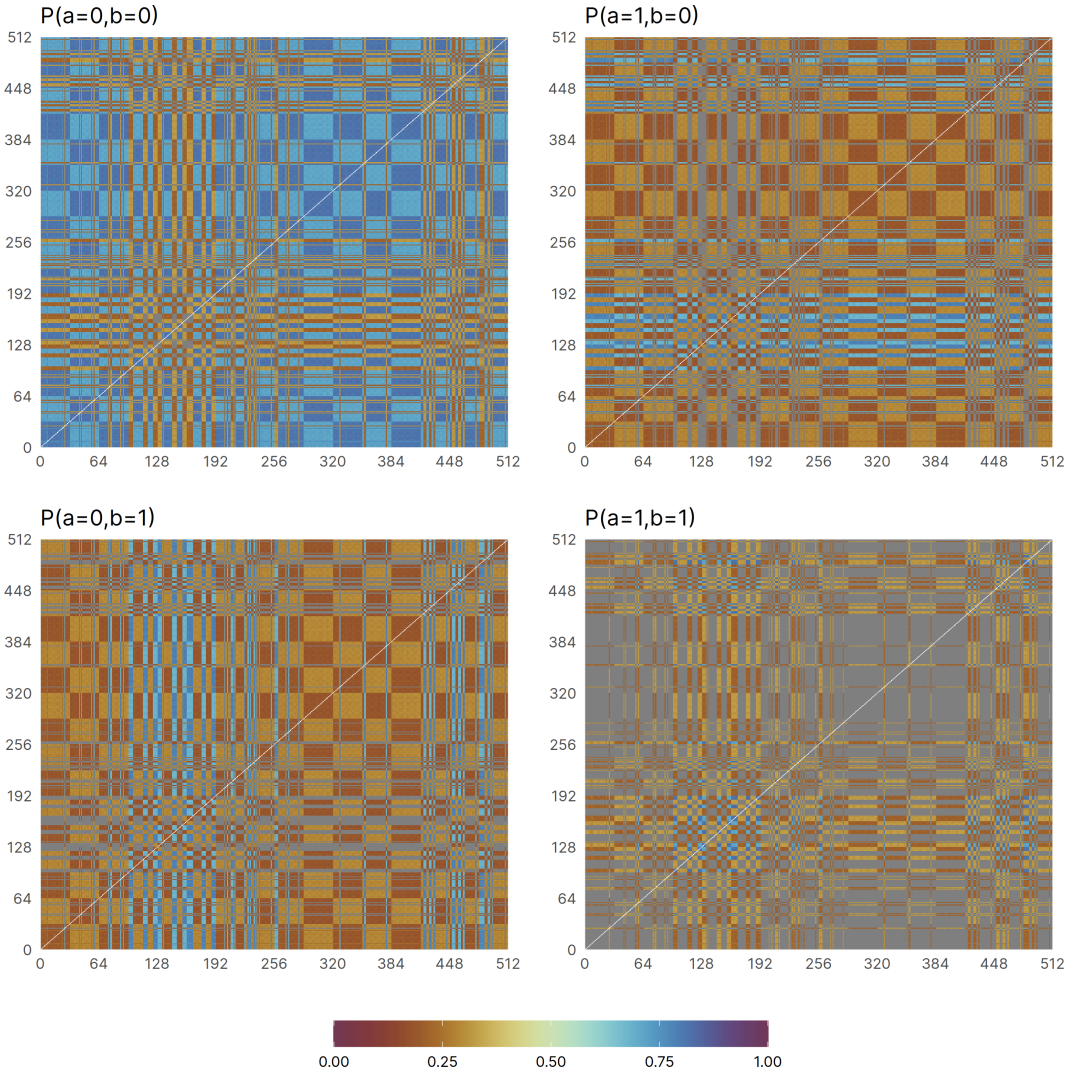


Figure 34: Average joint probabilities of the SRAM. The divergency has been computed per blocks of 512 bits and averaged out.

These spatial phenomena appear also when looking at the Reliability Invariance per block, as is show in Figure 35. We can see that only the handful of bits that don't present correlation phenomena present good Reliability, while the other bits tend to be unreliable or too reliable but Bit-aliased.

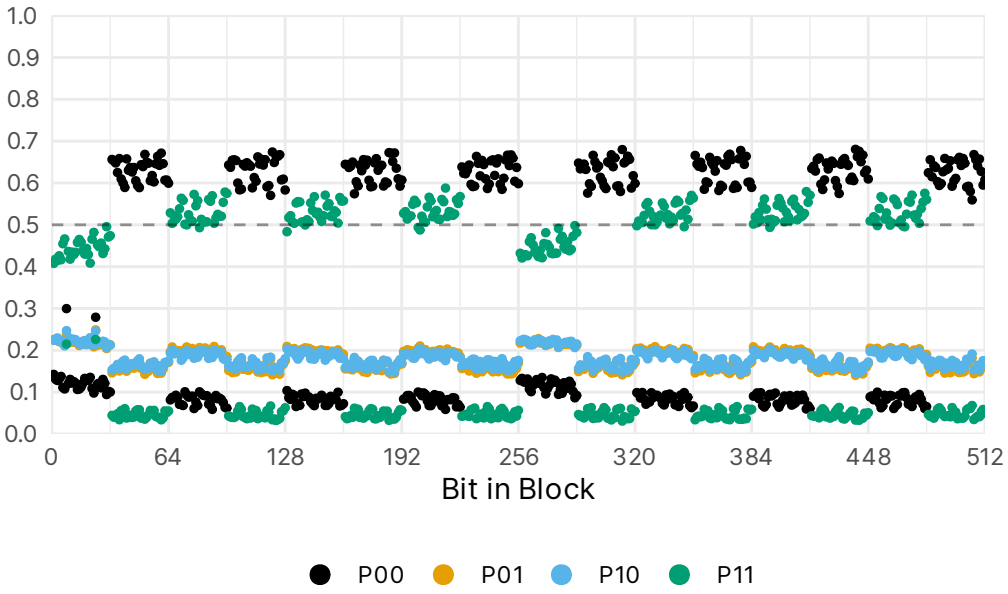


Figure 35: Reliability Invariance of the SRAM-PUF

Furthermore, the Punctual Bit-aliasing has been computed and its heatmap is showcased in Figure 36. Initially we see that the behaviour is similar to the one of Bit-aliasing. However, a clear cut occurs in the middle of the memory, which could be indicative of the use of a SRAM macro design that was repeated multiple times.

By taking the difference between the Bit-aliasing and Punctual Bit-Aliasing, the cut at the middle of the memory becomes even more apparent as showcased in Figure 37.

This cut is an indication that the SRAM was design most likely by using a macro that creates multiple copies of smaller individual memory blocks. This is not far from reality as SRAMs are quite complex to design and most times their design is automated using

As it was also proposed in Chapter 2, the Kullback-Leibler diverge of the Punctual Bit-aliasing and Bit-aliasing has also been computed and showcased in Figure 38. Again, there is a spike right in the middle of the memory (40kiB) and another one at 48 kiB as in Figure 30.

Recall that the divergence is always strictly non-negative. The negative values are due to challenges with extreme (i.e. 0 or 1) Bit-aliasing or Punctual Bit-aliasing.

By grouping the values obtained into blocks of 512 bits wide, we obtain the results showcased in Figure 39.

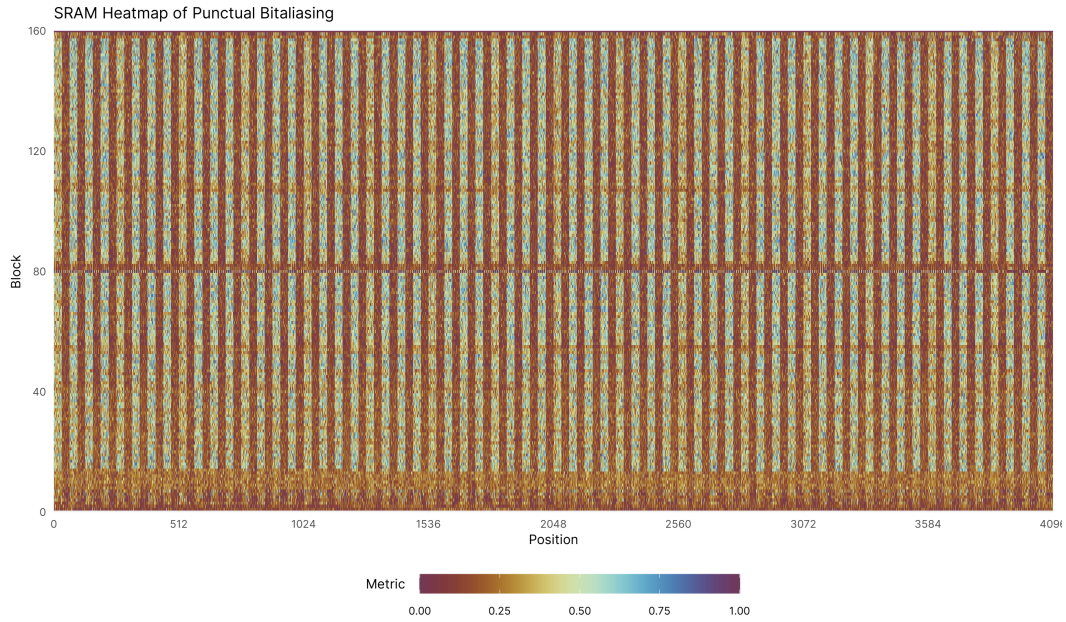


Figure 36: Heatmap of Punctual Bit-aliasing across all bits in the SRAM

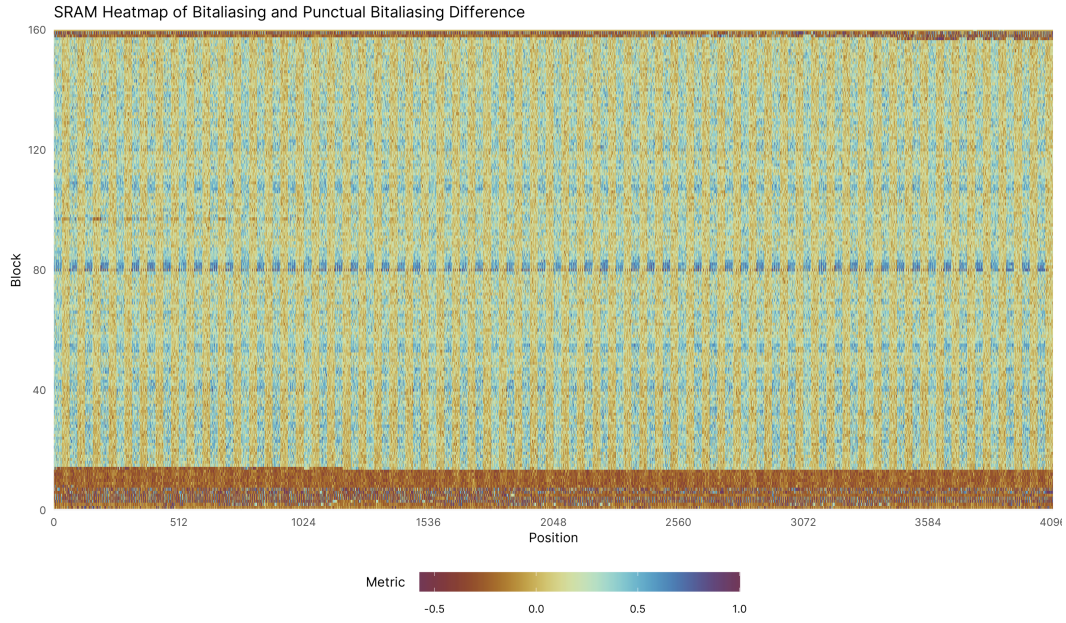


Figure 37: Heatmap of difference between Bit-aliasing and Punctual-Bitaliasing across all bits in the SRAM

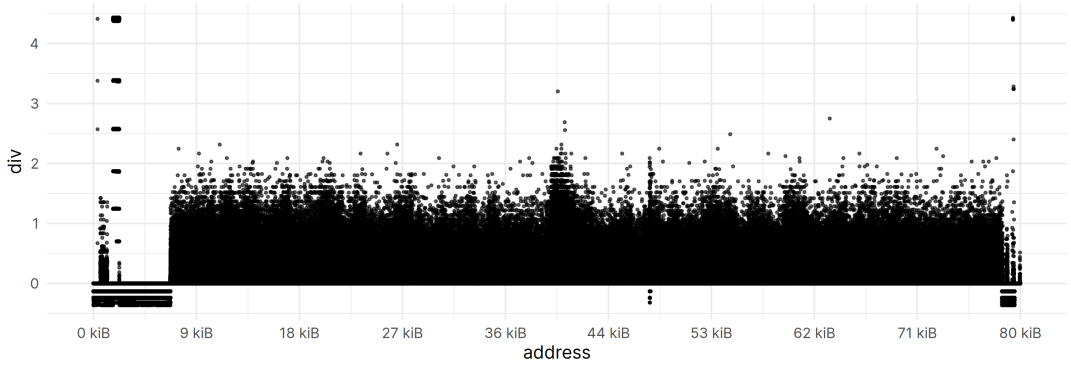


Figure 38: Kullback-Leibler divergence between Bit-aliasing and Punctual Bit-aliasing on the SRAM

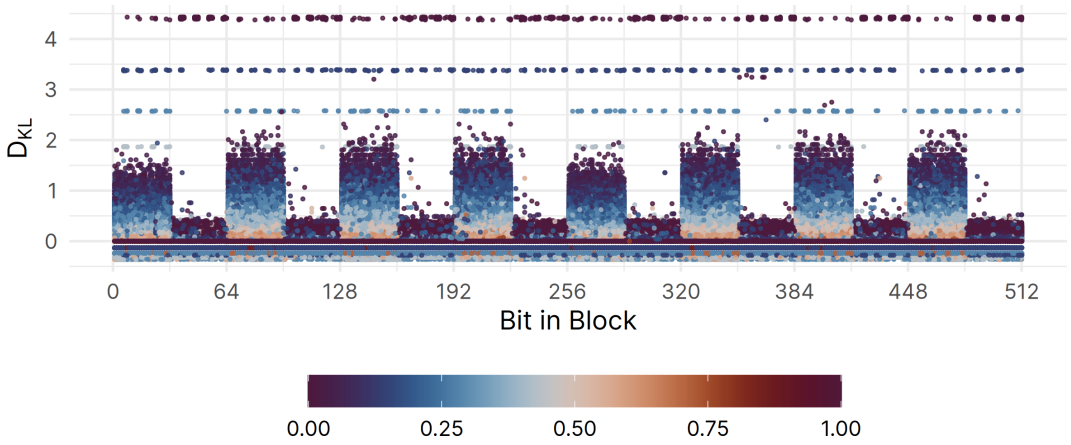


Figure 39: Average Kullback-Leibler divergence between Punctual Bit-aliasing and Bit-aliasing on each SRAM block.

We see that bits 8 and 24 in the first 32 bits have much lower divergence than the rest of the bits, which means that those bits are more reliable and represent the random behaviour.

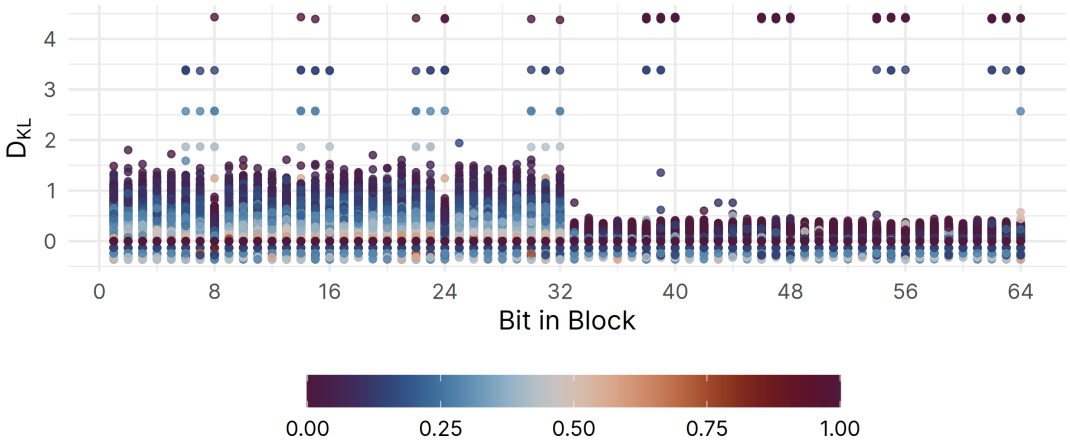


Figure 40: Zoomed view of the average Kullback-Leibler divergence between Punctual Bit-aliasing and Bit-aliasing on each SRAM block.

Another important analysis is performed with the Mutual information presented in Appendix A. Recal that the Mutual information quantifies how much we can learn about a response by knowing another one. This means that if we know information about some bits, we can reduce the entropy about other bits.

The different Figures Figure 41, Figure 42, Figure 43 correspond to block sizes of 512, 1024, 2048 bits respectively. Due to the amount of information condensed in these images, it's difficult to read them. The result of these analysis show that there are bits in "bad" areas of memory that share information with the "good" bits that are located at the beginning at the start of each 512 bits block. The diagonal represents the information gained from a bit given the same bit, so obviously the information gained is maximum. However, by looking closely at Figure 42, it's possible to discern two other diagonals. This represent the maximum information gained about some bits given a series of bits that are always at a certain offset. This information can have critical repercusions as an attacker could exploit this behaviour to extract information about protected memory areas by analysing areas with high correlation. While the information gained per bit is minimal, if multiple bits are gathered, an attacker could potentially carry out this attack. Further work will be performed to validate this attack.

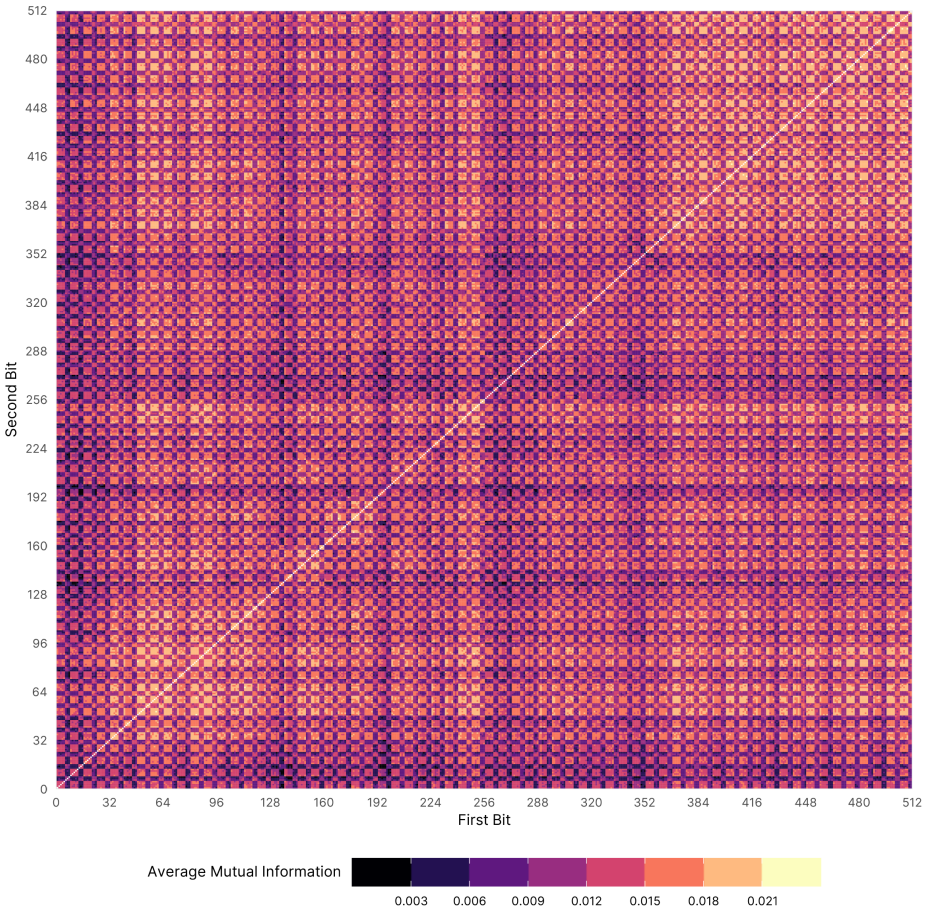


Figure 41: Average Mutual Information per block of 512 bits of the SRAM

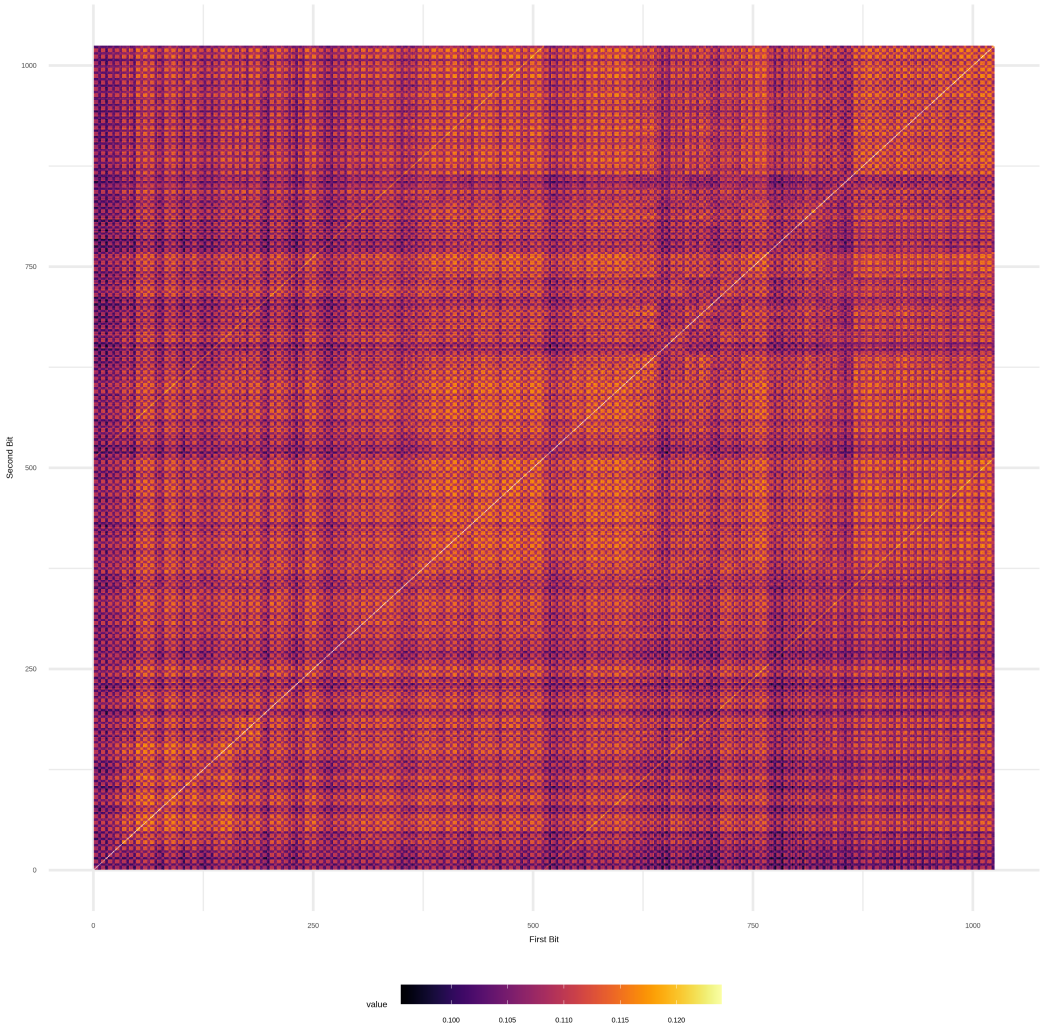


Figure 42: Average Mutual Information per block of 1024 bits of the SRAM

Figure 43: Average Mutual Information per block of 2048 bits of the SRAM

3.2 Ageing and NBTI Effects

In order to validate the NBTI studies performed on the SRAM boards, we analised the effects of the metrics, mainly Bit-aliasing, in time. The Bit-aliasing has been analyzed per sample across the two groups. Figure 44 showcases the Bit-aliasing of both the NBTI sample set and the reference one.

A weird effect can be seen. Every sample on the NBTI group contains a disruption in the distribution of Bit-aliasing that corresponds to the number of writes, (i.e. number of samples). That is to say, the second sample, gathered after one write operation has one valley at the center. The third sample, after two write operations, has two valleys that split the memory in three different sections. While this effect also appears in the first two samples of the reference group, the effect is much more apparent in future samples. To the best of our knowledge, this is the first time this behaviour is observed and analized in detail.

All these observed effects are highly indicative of being caused by the topology and design of the SRAM. For example, studies such as those by Barbareschi et al. [129] on STM32F3 and STM32F4 devices show no biasing patterns. However, we are not the only researchers to observe these biasing patterns. Studies by Masoumian[1] and Abideen[2] have also identified the same biasing patterns. The importance of layout and design techniques on PUF quality has already been studied and documented by Talukder [196] and Cortez [197]. Using simulation models and measurements, the authors have demonstrated that increasing the temperature leads to an increase in bias in SRAM PUF for the targeted designs. Specifically, the impact of ramp-up time on bias in SRAM PUF has been highlighted. To reduce the effect of temperature-induced bias, one could slow down the VDD ramp-up time. A slower ramp-up time results in Hamming weights closer to 0.5, thereby mitigating the impact of mismatches in the power supply network.

Further analysis of these devices is severely limited. We do not have access to the design or layout of the STM32 devices, nor does the current iteration of the SRAM platform or the devices themselves have a way to configure the ramp-up voltage. Future work will investigate the nature of these patterns by analyzing possible reasons related to different SRAM topologies.

The second proposed extension, Reliable Entropy, has been calculated to demonstrate its efficacy in identifying the appropriate CRPs, as depicted in Figure 45. It is not apparent but the count of identified acceptable CRPs is quite limited, which is comprehensive given the results displayed above. Nevertheless, employing this metric provides assurance that the identified CRPs are secure and reliable.

[1] S. Masoumian *et al.*, 'Modeling and analysis of sram puf bias patterns in 14nm and 7nm finfet technology nodes,' in *2023 IFIP/IEEE 31st International Conference on Very Large Scale Integration (VLSI-SoC)*, IEEE, 2023, pp. 1–6.

[2] Z. U. Abideen *et al.*, 'Impact of orientation on the bias of sram-based pufs,' *arXiv preprint arXiv:2308.06730*, 2023.

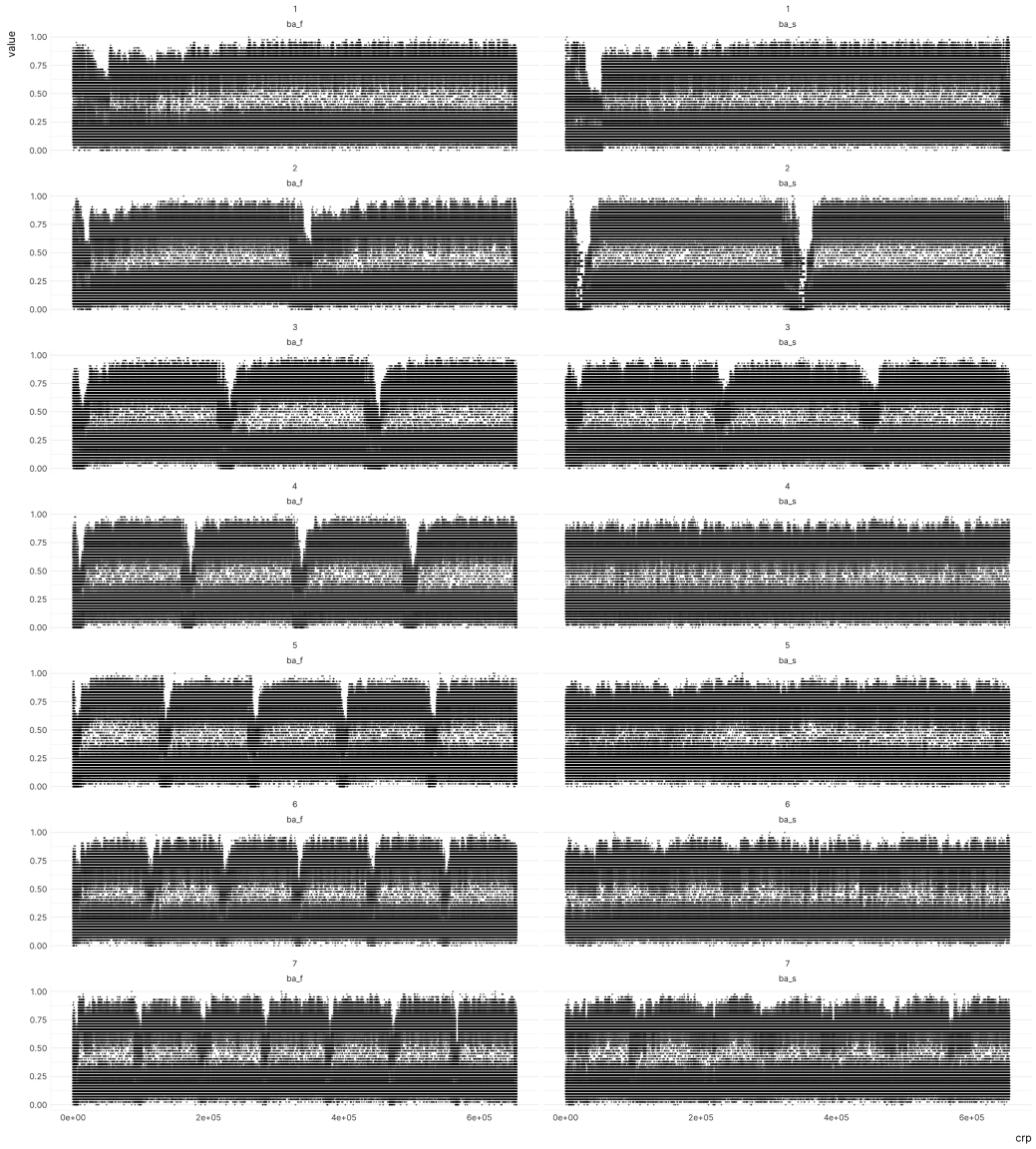


Figure 44: Scatterplot of the SRAM Bit-aliasing after different samples. Left represents the NBTI study group and right the control group.

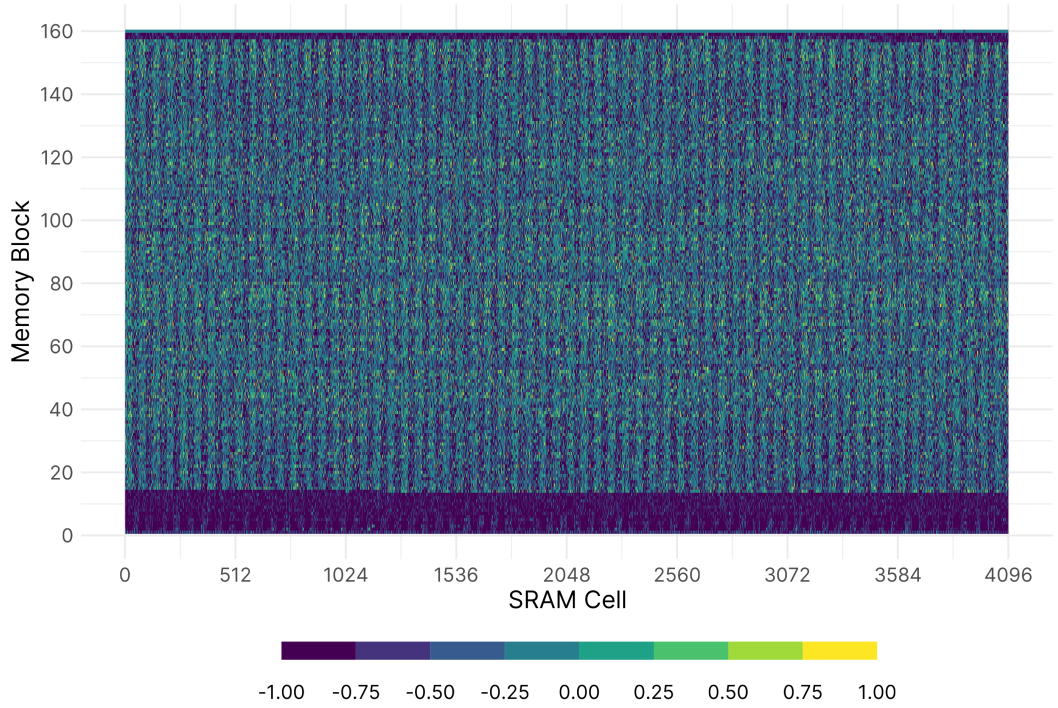


Figure 45: Heatmap of the Reliable Entropy of a single SRAM

3.3 Infineon RO

An initial evaluation of the different oscillation frequencies is provided. Recall from Chapter 3 that each device is composed of 6 blocks of 255 ROs.

We can see as the number of stages increases, the frequency becomes smaller and we have less variability, so the distribution is much narrower since the effects of variability average out.

The metrics have been also computed. They are summarised in Table 15 and are represented in Figure 47. It's evident that the Infineon RO-PUF behaves much better than the SRAM-PUF, as the distributions of the metrics are unimodal and centered around 0.5.

Table 15: Summary of the Infineon RO-PUF metrics

Metric	μ	σ	$CI_{95\%}$	[min, max]
Uniformity	0.5844	0.4928	[0.5834, 0.5854]	[0.2106, 0.8525]
Bit-aliasing	0.5844	0.4928	[0.5834, 0.5854]	[0.03258, 1.00000]
Uniqueness	0.4211	0.0951	[0.4204, 0.4217]	[0.09939, 0.83231]

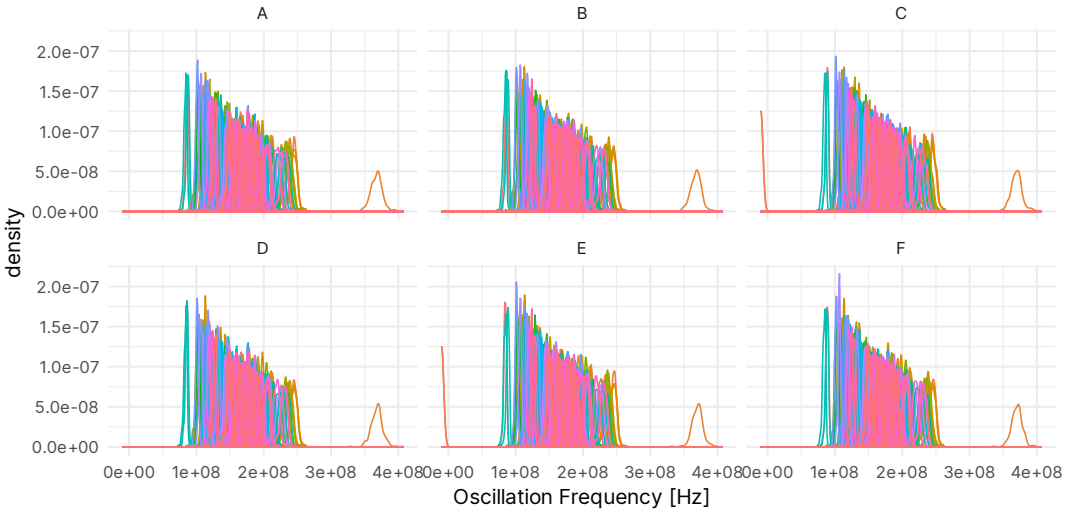
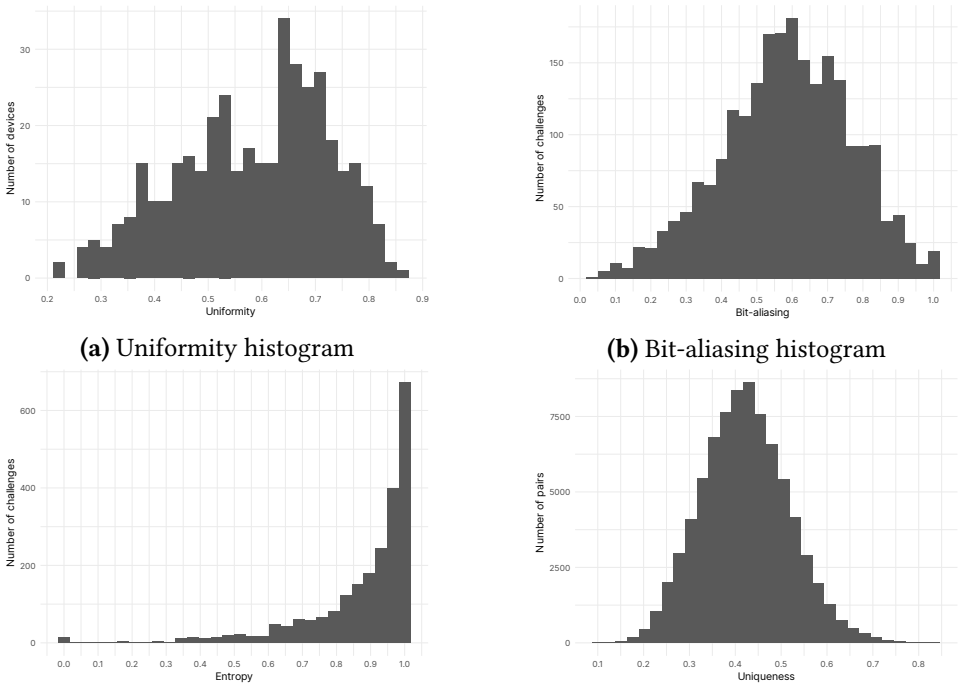


Figure 46: Distribution of RO frequencies in the 6 blocks of the Infineon circuit.



(a) Uniformity histogram

(b) Bit-aliasing histogram

(c) Entropy histogram

(d) Uniqueness histogram

Figure 47: Histogram of the Infineon RO-PUF metrics

The reliability Invariance has also been computed and is showcased in Figure 48. We see that most challenges have the expected behaviour and there are no asymetrics like in the SRAM.

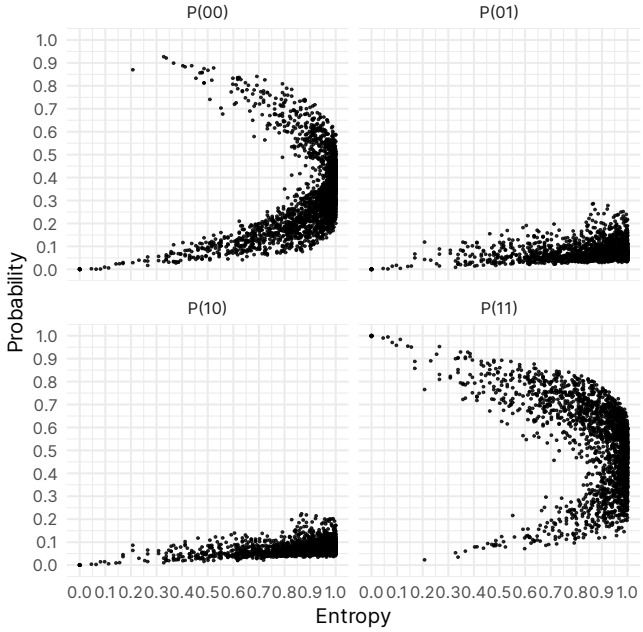


Figure 48: Reliability Invariance of Infineon RO-PUF

Analogous to the SRAM, the Punctual Bit-aliasing has also been evaluated. Since the RO-PUF performs very well and does not seem to have any correlation effects, the Punctual Bit-aliasing is very close to the reference Bit-aliasing. We can see that while most of the responses do not present any deviation from the canonical Bit-aliasing, responses with Bit-aliasing close to 0 do present high divergence which is due to the non-nil probabilities of bit flips across all values of entropy, shown in Figure 48.

The Reliable Entropy has not been computed on the RO-PUF as it behaves much better than the SRAM PUF. The filtering techniques for the RO-PUF exploit the techniques presented in Chapter 2.

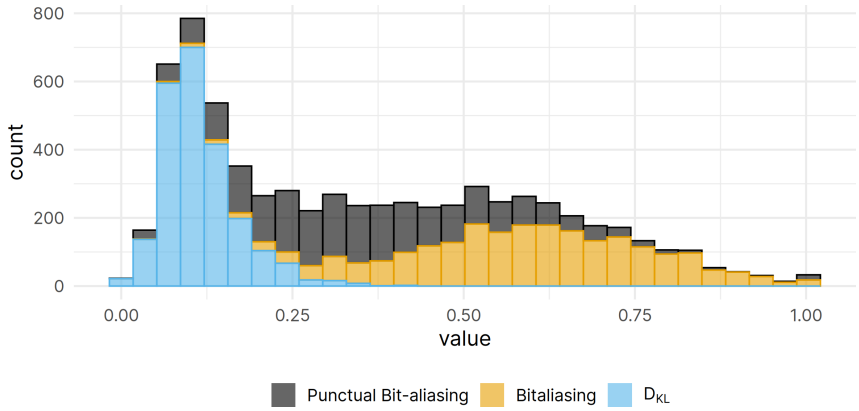


Figure 49: Kullback-Leibler divergence between Bit-aliasing and Punctual Bit-aliasing on the Infineon PUF.

SRAM Digraph

VI.4

In this section, as a novel analysis for PUFs, techniques from binary forensics will be employed. Given the strong correlation effects between SRAM cells, we can employ these techniques to visually inspect and analyse spatial correlations. One of such possible tools is the Digraph (not to be confused with Directed Graph), which was originally presented by Greg Conti at Blackhat in 2010 for the classification of file format based on the content. The idea is to study the relationship between contiguous bytes, which in this case allows studying relationships in address space. Each sequential byte pair corresponds to the coordinate of a 256×256 Cartesian graph. Once all pairs have been computed, the number of repetitions is tallied up.

Given a bytestring B, it's corresponding Digraph G is computed as shown in Equation 4.1, where the $\delta(a, b)$ function returns 1 if $a = b$ and 0 otherwise.

$$(G)_{x,y} = \sum_{i=1}^{n(B)-1} \delta\left(B_i, x\right) \cdot \delta\left(B_{i+1}, y\right) \quad \{x, y \in [0, 255]\} \tag{4.1}$$

The Python code 4.1 shows the Digraph computation for a random vector of bytes.

```
import numpy as np

digraph = np.zeros((256, 256))

sram_bytes = [0, 10, 255, 128, ...]

for i in range(length(sram_bytes) - 1):
    first = sram_bytes[i]
    second = sram_bytes[i+1]
    digraph[first, second] += 1

# A visual aid of how the pairs are computed
#
# 000 010 255 128
# ---------------
# 000 010
#     010 255
#         255 128
```

Listing 4.1: Digraph computation using Python.

For each device out of the 84 available, it's corresponding Digraph has been computed. The results have been averaged out into a single Digraph shown in Figure 50. The tally has been represented in the logarithmic scale due to the disproportionate amount of 0s that appear in the stack of the memory. Notice in the figure that the diagonal separatrix does not indicate symmetry, although an even distribution of counts across both regions of the Digraph is indicative of an even distribution of bytes in the memory.

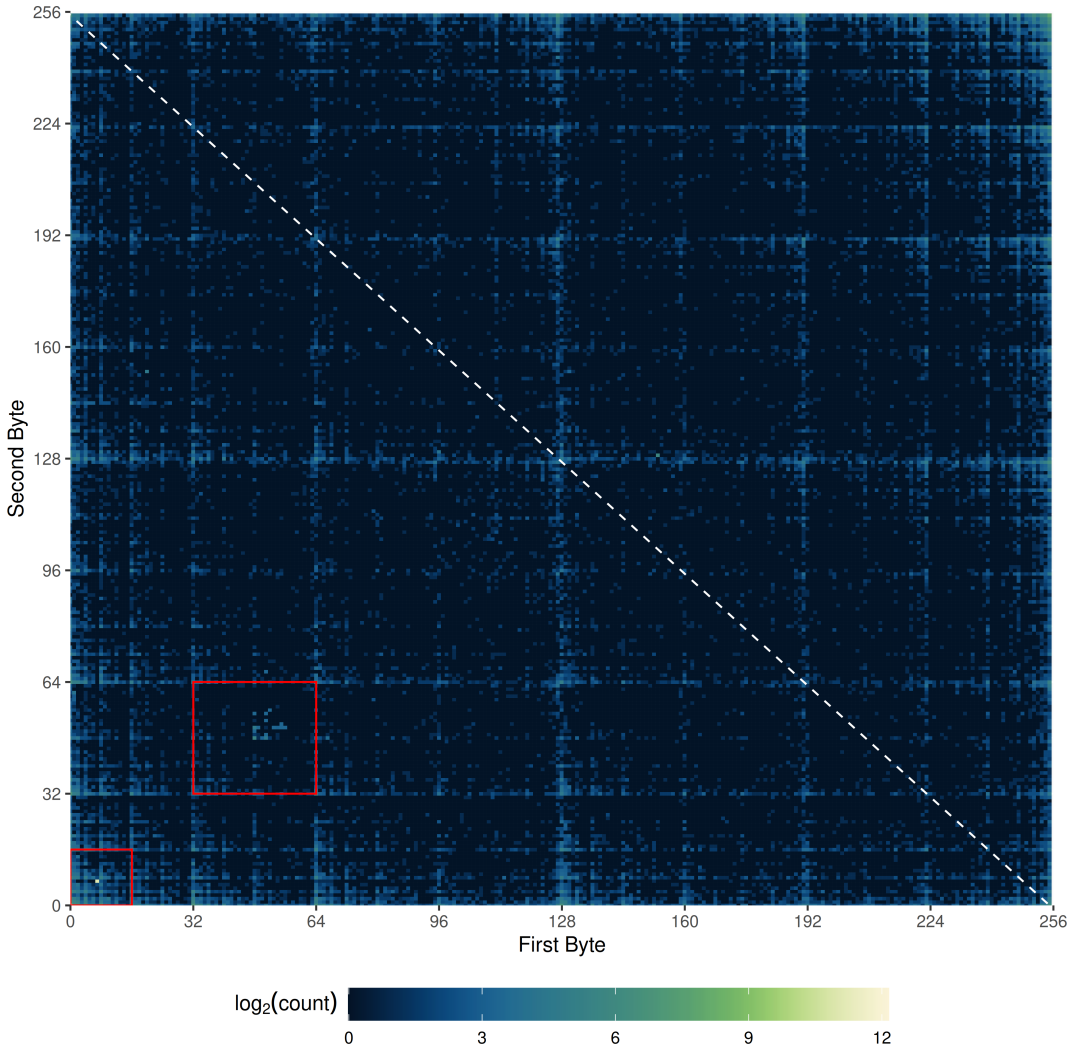


Figure 50: Digraph of the SRAM under study

The fractal-like pattern that arises is due to the heavy complementing biasing effects that have already been presented. At first glance, it's evident most of the values appear in the bottom left corner, which corresponds to low byte values. The stack was used in the computation and recall that these regions are mostly filled with 0s. Moreover, at position $(48, 48)$ a pattern appears that does not exist on the mirrored position.

Individual byte probabilities can be computed through the aggregation of column counts. Here, b_i represents the i-th byte in the bytestring and it's corresponding Digraph G has been normalized. In that case, each individual position corresponds to the conditional probability depicted in Equation 4.2.

$$(G)_{xy} = P(b_i = x \mid b_{i+1} = y) \tag{4.2}$$

By averaging up all values in a column, we obtain the probability of finding the byte corresponding to that column as depicted in Equation 4.3.

$$P(b = x) = \frac{1}{255} \bigcup_{y=0}^{255} P(b_i = x \mid b_{i+1} = y) \tag{4.3}$$

This procedure has been performed on the SRAM dataset and the results are showcased in Figure 51. Additionally, each byte position is coloured depending on it's Hamming Weight.

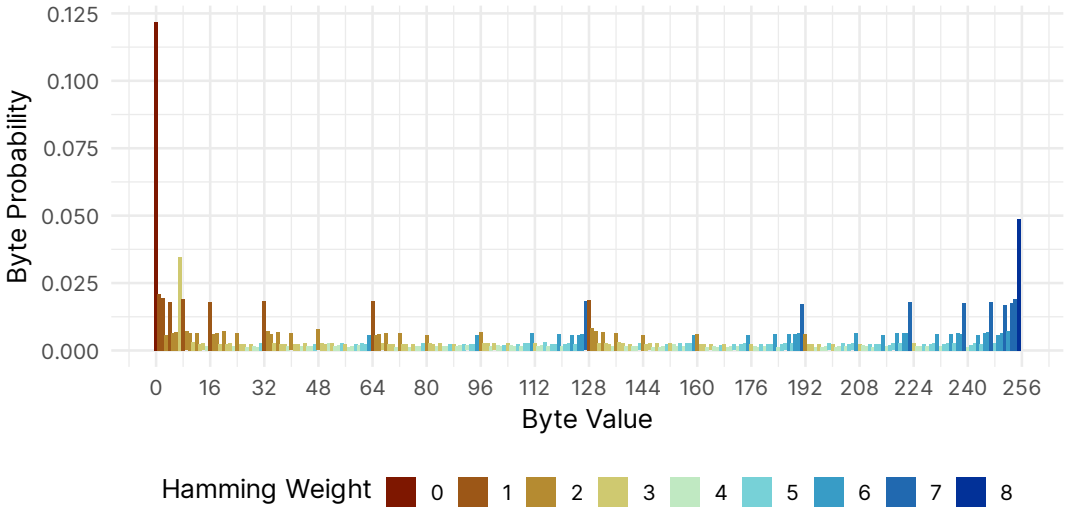


Figure 51: Byte value probabilities based on the column average of the Digraph

It is evident from the figure that the distribution of byte values is not uniform. However, as said above, the distribution does not need to be uniform to have an even distribution of responses in the SRAM, it just needs to be symmetric around the separatrix, which in this case corresponds to the median of 255.

By analysing the more prevalent values, an interesting pattern appears in the values. The list of more common values along with their binary representation is portrayed in Table 16. Notice that values higher than 127 are the complement of those lower than 127. Moreover, values lower than 127 have a Hamming Weight of 1, and 7 if higher than 127. We can also use this data to corroborate that the SRAM is biased towards 0, as most values are gathered around the value 16. This is most likely due to the implementation of the SRAM and no further analysis can be performed without more information about the design and the implementation.

Byte	Binary representation
000	00000000
008	00001000
016	00010000
032	00100000
064	01000000
127	01111111
128	10000000
191	10111111
223	11011111
239	11101111
247	11110111
255	11111111

Table 16: Common values in the digraph and their binary representation.

Following to this, the probabilities depicted in Figure 51 are aggregated by Hamming Weight and the result is displayed in Figure 52. Notice that the behaviour is symmetric when taking complementary pairs of weights (e.g. 0 to 4, 1 to 5, 2 to 6, 3 to 7). Since the Hamming Weight 8 only corresponds to the byte value 255, its behaviour is negligible. Again, the exceeding amount of values with Hamming Weight of 3 is also present in the graph, by comparing the aggregated probabilities of 3 and 7.

Due to this unique distribution of values, the analysis is extended to study the distribution of Hamming weights. Each byte it's converted to it's corresponding Hamming Weight. The resulting Hamming Weight Digraph shown in Figure 53 is now a Cartesian plane of 8×8.

There are a total of $\binom{8}{3} = 56$ possible byte values with Hamming Weight of 3, which represent 22% of the total possible values as represented in Table 17.

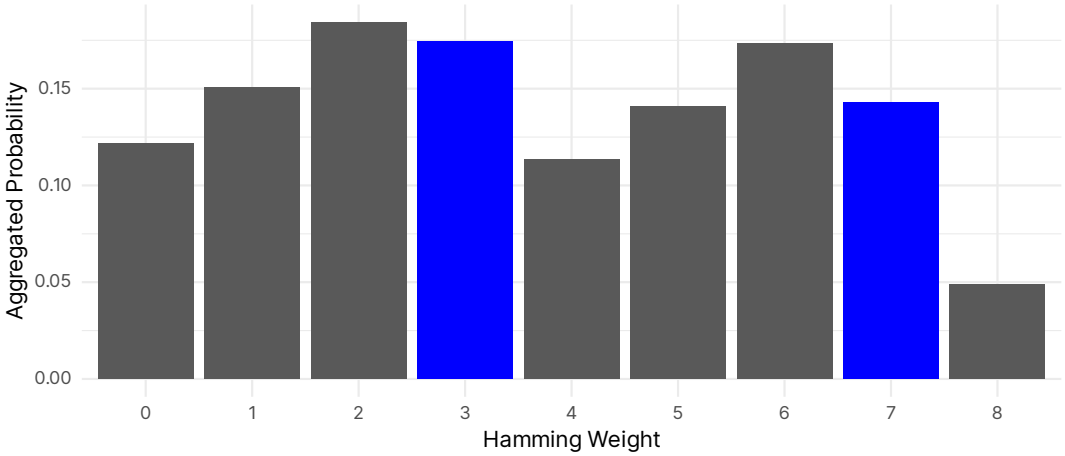


Figure 52: Hamming Weight probabilities based on the column average of the Digraph

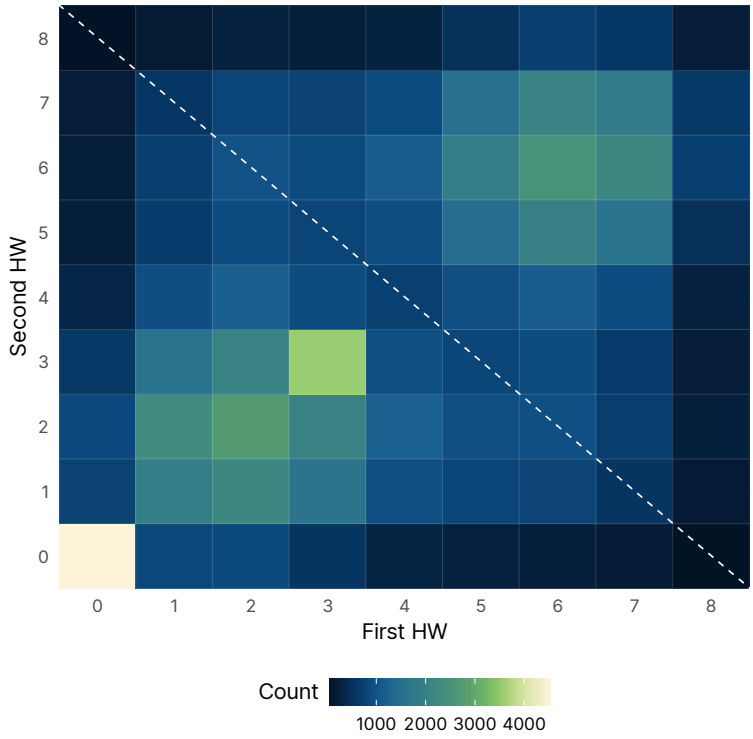


Figure 53: Hamming Weight digraph of the SRAM under study

Table 17: Number of values and ratio per Hamming Weight

Hamming Weight	Number of values	Ratio (Number / 256)
0	1	0.00
1	8	0.03
2	28	0.11
3	56	0.22
4	70	0.27
5	56	0.22
6	28	0.11
7	8	0.03
8	1	0.00

As represented in Figure 54, most of the values are in the left part of the graph, which reassert the skewness of the memory values and by superposing the first obtained digraph, most of the values that occur in the memory with a Hamming Weight of 3 are between 32 and 64.

Without any information about the layout and physical implementation of the memory, further work is hindered. The Digraph and other general tools similar to this one could be added to the set of PUF metrics to assess for correlation effects. Another interesting extension would be to study if the transition probabilities derived from Figure 50 are sensitive to spatial phenomena, wherein the probabilities not only depend on the subsequent bytes but also the address they are located, although the results shown so far would indicate otherwise.

4.1 Extension to Markov Chain

We can extend this notion of Digraph in order to compute the transition matrix of a Markov Chain that could model the sequential behaviour of the memory. We are interested in computing the probability of the next byte given the current byte, as opposed to the computation of the Digraph.

$$(P)_{x,y} = P(b_{i+1} = x \mid b_i = y)$$

We can see from the Digraph definition in Equation 4.2, that the transpose of the normalized Digraph corresponds to the right stochastic matrix we are looking for.

$$(G)_{xy} = P(b_i = x \mid b_{i+1} = y) \Longrightarrow (G^T)_{x,y} = P(b_i = x \mid b_{i+1} = y) \tag{4.4}$$

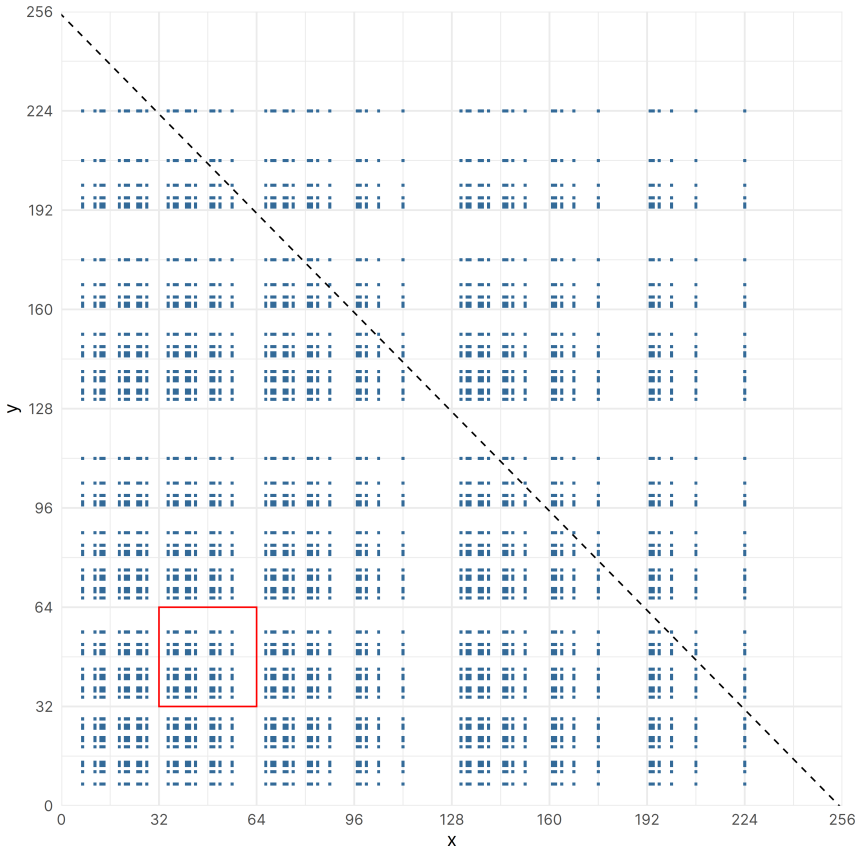


Figure 54: Digraph containing only pairs with Hamming Weight of 3

Conclusions

VI.5

As shown in previous chapters, the canonical metrics strive to measure critical aspects of PUFs, notably Reliability and Entropy. The connection between the Reliability and Entropy of PUFs was studied in detail since understanding this relationship is crucial for enhancing the performance and security of PUFs in various applications.

In the context of RO-PUFs, an initial study examining the distribution of frequency differences and reliability was examined. It is commonly agreed in the literature that low frequency differences can be deemed unreliable, but it is challenging to establish a numerical threshold. To systematically analyse this, a simulation-based offline methodology was proposed. This methodology, which does not require extensive real-time testing, computes the bounds necessary to distinguish between reliable and unreliable responses. Furthermore, this methodology can be extended to other differential PUFs, where 2 nominally identical components are compared.

Building on this relationship, a novel model called "Time-To-Response" was introduced. Although this information can be numerically computed from experimentally derived data, the proposed model provides a deeper understanding of how the frequency difference distribution influences the time required to obtain a desired number of reliable responses. By relating these 2 factors, this model helps in predicting the PUF's performance over extended periods and under different environmental conditions. This approach offers a practical way to assess and improve the reliability of PUFs in real-world applications.

Expanding further, the relationship between frequency difference and entropy yield on RO-PUFs was also investigated. While the connection between frequency difference and reliability is relatively well-studied, it's impact on entropy is less understood. Entropy is a critical measure of a PUF's uniqueness and security, as higher entropy indicates greater unpredictability and resistance to attacks. The performed analysis suggests that frequency difference can also be a predictor of the entropy yielded by a PUF. This hypothesis was initially based on simulations conducted for the reliability study and is subsequently validated using the extensive Infineon RO dataset. These models were comprehensively analysed and validated using both simulation results and real data. Indeed, the simulation and experimental results demonstrate how simulation-based methodologies and real-world data can be integrated to provide a holistic understanding of PUF performance.

In conclusion, a detailed exploration of the connection between Reliability and Entropy, a series of new statistical models and methodologies for their analysis were provided in this chapter. The integration of simulation results with real-world data highlights the critical role of empirical validation in advancing PUF technology.

VII

PUF Design Proposal

Split PUF

VII.1

We have already shown that both Entropy and Reliability play a huge role when it comes to PUF adoption. However, most of the proposed techniques require a great knowledge of the target technology, or expensive solutions.

The design methodology proposed in this chapter is focused on RO-PUFs. This approach can be generalized to other families of PUFs, provided that the response is computed by comparing two nominally identical instances of a component. The core concept is the comparison of two elements positioned identically across two separate dies.

This solution greatly simplifies the design process. Remarkably, it eliminates the need for specialized design knowledge or an understanding of security or PUF intricacies. The only requirement is to design a set of entropy sources, utilizing half the number of ROs compared to the traditional RO-PUF approach. This efficiency is achieved without compromising the integrity of the entropy source. A key advantage of this methodology is that it addresses the primary issue highlighted in the previous section—bias in the challenge. However, it does introduce a new challenge: bias in the device itself. This bias arises if one die exhibits a slightly higher voltage than the other, causing all ROs on the first die to be faster than those on the second. Consequently, this could result in a uniform distribution of responses.

To mitigate this problem, we introduce a pre-processing operation that ensures the average frequencies of the ROs on both dies are aligned. The process for reading PUF responses involves several steps: initially, the raw frequency values of the ROs are read from both dies. Then, statistical distributions of these frequencies are created, capturing the mean and standard deviation of each die. The frequencies are then corrected by aligning them to have the same mean and sigma. The correction is performed using Equation 1.1.

$$f_{corrected} = \frac{(f_{raw} - \mu_{raw})}{\sigma_{raw}} \cdot \sigma_{target} + \mu_{target} \tag{1.1}$$

Where $f_{corrected}$ is the corrected frequency, f_{raw} is the raw frequency, μ_{raw} and σ_{raw} are the mean and standard deviation of the raw frequencies, and μ_{target} and σ_{target} are the target mean and standard deviation.

With this correction applied, the frequencies from the two dies are bit-wise compared to generate the final PUF response. This alignment ensures that any inherent device biases are neutralized, thereby maintaining the integrity and reliability of the PUF responses.

1.1 Proposed method

In part IV, we have proposed that each challenge might have a bias due to systematic variability and design choices (being a challenge the comparison of two elements in the same positions for all devices), which we call "bias in the challenge". Moreover, we propose the idea that each device might have a bias due to operating conditions (which might fluctuates from one device to another), which we call "bias in the device."

Initially, we design all ROs to oscillate at the desired oscillating frequency f. Due to process variability and design considerations, the ROs in the PUF oscillate following a frequency distribution

$$F \sim N(\mu, \sigma)$$

Where μ is chosen at design and depends on the number of RO stages and σ depends on the process variability and design choices. A single device d composed of N ROs has a series of frequencies that follow the frequency distribution F

$$F_d = \begin{bmatrix} f_1 & f_2 & f_3 & \cdots & f_N \end{bmatrix}$$

The total system composed of D devices each one with N Ring Oscillators can be represented in a matrix as follows

$$F_{D,N} = \begin{bmatrix} f_{1,1} & f_{1,2} & \cdots & f_{1,N} \\ f_{2,1} & f_{2,2} & \cdots & f_{2,N} \\ \vdots & \vdots & \vdots & \vdots \\ f_{D,1} & f_{D,2} & \cdots & f_{D,N} \end{bmatrix} \tag{1.2}$$

Due to the different lots in manufacture and process variability, each device is subjected to some *Shift* $\Omega \sim N(0, \sigma_\Omega)$ that displaces every frequency.

$$F_d = \begin{bmatrix} f_1 + \omega_d & f_2 + \omega_d & \cdots & f_N + \omega_d \end{bmatrix}$$

Because of this, each device is represented as the original frequency distribution plus the Shift

$$F_d \sim N(\mu, \sigma) + \Omega$$

Since the Shift is normally distributed and centred around 0, the mean frequency distribution does not change. However, the variance of the frequency distribution increases in the following manner

$$\mathrm{E}[F_d] = \mathrm{E}[F_d] + \mathrm{E}[\Omega] = \mathrm{E}[F_d] \tag{1.3}$$
$$\mathrm{Var}[F_d] = \mathrm{Var}[F_d] + Var\Omega \tag{1.4}$$

Moreover, each RO at every position is subjected to some Bias $B \sim N(0, \sigma_B)$

$$F_n \sim N(\mu, \sigma) + B$$

$$F_n = \begin{bmatrix} f_{1,n} + \beta_n \\ f_{2,n} + \beta_n \\ \vdots \\ f_{D,n} + \beta_n \end{bmatrix}$$

We can also add a term ε to account for environmental changes where $E \sim N(0, \sigma_\varepsilon)$. That means that the frequency of a RO chosen at random is represented by Equation 1.5

$$f = \mu + \omega_d + \beta_n + \varepsilon_t \tag{1.5}$$

There are multiple architectures of ROs that this analysis could be applied to. For the shake of simplicity we are going to keep using the same design that was used for the simulations and analysis during this thesis. If we split the N Ring Oscillators into 2 equal groups and compare each RO of each group to every other RO of the other group the number of frequency differences, and thus responses is given by:

$$\text{Number of Responses} \equiv C = \frac{N}{2} \times \frac{N}{2} = \frac{N^2}{4}$$

In a device d, a single frequency difference c is computed as follows, using as an example ROs 1 and 2.

$$f_{\Delta d,c} = f_{d,1} - f_{d,2} \tag{1.6}$$
$$= (f_1 + \omega_d + \beta_1) - (f_2 + \omega_d + \beta_2) \tag{1.7}$$
$$= (f_1 + \beta_1) - (f_2 + \beta_2) \tag{1.8}$$
$$= (f_1 - f_2) + (\beta_1 - \beta_2) \tag{1.9}$$

The following equations summarize the expected value and variance of the frequency difference distribution by taking into account all the bias.

$$\mathrm{E}\left[F_{\Delta d}\right] = \mathrm{E}\left[F_{\Delta d}\right] - \mathrm{E}\left[F_{\Delta d}\right] + \mathrm{E}\left[B\right] - \mathrm{E}\left[B\right] \approx 0 \tag{1.10}$$

$$\mathrm{Var}\left[F_{\Delta d}\right] = \mathrm{Var}\left[F_{\Delta d}\right] + \mathrm{Var}\left[F_{\Delta d}\right] + \mathrm{Var}\left[B\right] + \mathrm{Var}\left[B\right] \approx 2\sigma^2 + 2\sigma_B^2 \tag{1.11}$$

It's clear that the variance of the Frequency Difference Distribution with Bias is larger than the Variance of the Frequency Difference Distribution without Bias

$$\mathrm{Var}\left[F_\Delta + Bias\right] > \mathrm{Var}\left[F_\Delta d\right]$$

We study the relationship between μ, σ, Bit-aliasing to prove this claim. Figure 55 shows the relationship between μ_Δ, σ_Δ and Entropy and as we can see, for a given μ_Δ the higher the standard deviation the better the entropy.

$$\text{Bit-aliasing} = 1 - \Phi\left(\frac{0-\mu}{\sigma}\right) = 1 - \Phi(-Z)$$

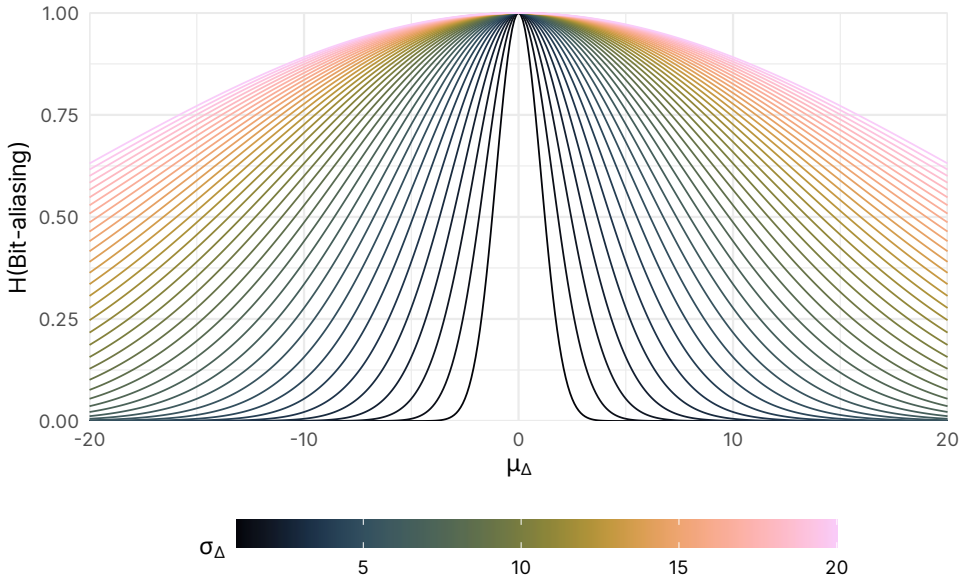


Figure 55: Relationship between μ_Δ, σ_Δ and Entropy

We can see from the figures that for any of μ the higher the σ the better the Bit-aliasing. This is due to the fact that wider distributions have more chances to have areas that cross the 0, no matter the value of μ. The ideal value is obtained when $\mu = 0$ as no matter the value of σ the distribution is always symmetric.

That means that in order to get the best Bit-aliasing we can either ensure that μ is as close as possible to 0, which is difficult to achieve in practice μ or to make sure that σ is as large as possible, for example by increasing the number of RO stages. From the model proposed,

we derive that the best Variance possible we can achieve to enhance the Bit-aliasing is the following

$$\text{Var}\left[F_{\Delta}\right] + \text{Var}\left[Shift\right] + \text{Var}\left[Bias\right]$$

1.2 Effect on the metrics

While this analysis is proposed here, further work will be needed to evaluate and prove this model, which due to time constraints, won't be performed here.

$$F_{\Delta D \times C} = \begin{bmatrix} f_{\Delta 1,1} & f_{\Delta 1,2} & \cdots & f_{\Delta 1,C} \\ f_{\Delta 2,1} & f_{\Delta 2,2} & \cdots & f_{\Delta 2,C} \\ \vdots & \vdots & \vdots & \vdots \\ f_{\Delta D,1} & f_{\Delta_{D,2}} & \cdots & f_{\Delta D,C} \end{bmatrix}$$

The objective will be to obtain a direct relationship between the different shifting parameters proposed above and their effect on the metrics, specially Uniformity and Bit-aliasing.

For Uniformity, we proposed the following model.

$$\text{Uniformity} = \begin{bmatrix} U_1 + \varphi_1 \\ U_2 + \varphi_2 \\ \vdots \\ U_D + \varphi_D \end{bmatrix}$$

$$\text{Uniformity} \sim B(\alpha, \beta) + \Phi \equiv B(\alpha, \beta) + B(\alpha_{\Phi}, \beta_{\Phi})$$

While for Bit-aliasing, an analogous model is proposed.

$$\text{Bit-aliasing} = \begin{bmatrix} p_1 + \pi_1 & p_2 + \pi_2 & \cdots & p_C + \pi_C \end{bmatrix}$$

$$\text{Bit-aliasing} \sim B(\alpha, \beta) + \Pi \equiv B(\alpha, \beta) + B(\alpha_{\Pi}, \beta_{\Pi})$$

1.3 Conclusions

As it was proposed by the model, the possible ways to increase the Entropy of the RO-PUF is to make the average frequency difference close to 0, which most of the time is extremely difficult or to increase the variance of the frequency difference distribution. Here we propose a design methodology specifically for PUFs that rely on comparing two nominally identical randomness sources. The methodology relies on splitting the randomness sources across different chiplets in order to maximize the variance of the distribution of the parameters of interest as a way to improve the performance of the PUF.

Work is in the making to assess this method on real data, but there is evidence on the validity of the Split PUF based on the model and already computed results.

This methodology however poses a security problem. The two independent halves can be more easily attacked independently, (for instance increase the temperature on one side, and not on the other side). Nevertheless, it is possible to embed a sensor (clearly in both halves, since they are duplicated) and the can easily detect an anomaly. Even in case of power-off attacks, the post-processing techniques can either tolerate the attack, or detect the anomaly (for instance the average of the frequencies of the two blocks are too far one from the other).

Conclusions

VII.2

This chapter has extended the previously discussed models and results to develop a comprehensive and holistic model aimed at significantly enhancing the performance of RO-PUFs. The development of this holistic model is a critical step in addressing the complexities and variabilities inherent in RO-PUFs, ensuring a more accurate and reliable framework for PUF analysis and design.

A new design methodology, coined “Split PUF”, provides many advantages over many solutions proposed in the literature to enhance the Entropy yield of RO-based PUFs. This new methodology relies on measuring nominally identical randomness sources positionally identically across different physical dies. Furthermore, a pre-processing stage is proposed in order to eliminate any bias across the different dies that may impact the behaviour of the PUF. By integrating the holistic model with the previously established relationship between Reliability and Entropy, a method to maximize the performance of RO-PUFs was devised.

Further work will be performed to evaluate the methodology proposed on real designs as well as exploring how this methodology can be adapted to other PUF families.

VIII

PUF Modeling

State of the Art

VIII.1

Mathematical modeling of PUFs is crucial for understanding their behaviour and enhancing their security features. The field of PUFs is rapidly advancing, characterized by an ongoing race between researchers and attackers. Machine Learning (ML) attacks on strong PUFs leverage advanced algorithms to predict responses, while cloning attacks on weak PUFs focus on physically replicating the PUF's unique properties. Despite these challenges, ongoing advancements in PUF design and defensive strategies aim to mitigate these threats, ensuring PUFs remain a robust tool in the security landscape.

Mathematical models not only help in approximating the behaviour of PUFs but also justify the randomness of the device, akin to "stochastic models" of True Random Number Generators and other security primitives. As highlighted before, PUFs offer numerous alternatives to traditional security primitives. However, the susceptibility of PUFs to various modeling and cloning attacks remains a critical area of research. This chapter delves into the current state of modeling techniques aimed at compromising PUFs, focusing on ML attacks on strong PUFs, cloning attacks on weak PUFs, and the mathematical modeling of weak PUFs.

1.1 Machine Learning-based approaches

Advanced ML methods have become central to statistical modeling across various scientific domains due to their high accuracy and ease of use. Strong PUFs, designed to generate numerous CRPs, are particularly susceptible to ML attacks. These attacks model the PUF's behaviour to predict responses to previously unseen challenges accurately.

Statistical analysis methods such as Support Vector Machines (SVM) and Particle Swarm Optimization [198] have proven effective in modeling PUF responses. Advanced ML approaches, especially deep learning techniques like Convolutional Neural Networks (CNNs), have shown remarkable success in capturing the intricate dependencies of PUF responses, particularly in strong PUFs. Numerous studies demonstrate the accuracy of these ML methods across various PUF designs [199, 200, 201, 202].

These methods not only predict responses but also identify potential weaknesses that could be exploited in attacks. Adversarial machine learning techniques further enhance this capability by simulating attack scenarios, allowing designers to develop more robust and tamper-resistant PUF architectures. By training algorithms on known CRPs, researchers can evaluate the susceptibility of PUFs to predictive attacks and refine PUF designs to enhance security. Side-channel attacks can also complement ML attacks, posing serious threats as demonstrated in [203] and [204]. For instance, [205] showed that techniques like Challenge Obfuscation can be easily compromised via power side-channel attacks.

Unfortunately, proposing solutions to ML-based attacks is challenging due to the difficulty in formally proving a design's robustness against statistical modeling. Works like [206, 207] have attempted to provide formal analyses on the "learnability" of PUF designs. Tools like PUFmeter [208] offer property testing for assessing PUF robustness against ML attacks. Assessment methodologies to measure the resilience of PUF designs against ML attacks have also been proposed [209, 210, 211, 212].

An important study is presented in [213], where the authors utilize BTF theory with Chow parameters to mathematically derive the min-entropy of the loop-PUF. They suggest that although the entropy of the loop-PUF increases quadratically with the number of stages, indicating strong resistance to machine learning attacks, the min-entropy and collision rate grow much more slowly, which implies that the resistance to cloning attacks may not be as robust as anticipated. This work underscores the notion that while a design may be highly effective against one type of attack, it may be more vulnerable to another.

Traditional techniques, such as creating confusion through XOR gates, have been proven ineffective [214, 215]. Innovative designs that aim to confound ML algorithms, like those proposed in [216, 217, 218, 219], offer potential solutions. The most notable design is the Interpose PUF [216]. However, some attacks have successfully compromised these defences [220] and [221]. Introducing specially crafted noise patterns can deter attackers by disrupting ML classification, though this requires additional systems to correct for the added noise [222]. The effects of ageing on PUF modeling are also crucial. Ageing misalignment between the training and attacking phases can hinder attack effectiveness [223]. Other proposals to hinder learning attacks include non-monotonic response quantization [224], where the quantized decision depends not only on the relative speed of paths but also on the distance between signal arrivals. Efforts to create "reconfigurable" designs that can modify the PUF itself [225] are promising but difficult to validate. Moreover, certain randomness sources, like Photonic PUFs, exhibit good resilience against ML and side-channel attacks [212], although they are highly sensitive to environmental conditions.

Overall, designing robust defences against ML attacks is highly challenging due to the rapid advancements in ML and statistical methods. Continued research and innovation are necessary to develop effective countermeasures and ensure the security of PUF-based systems.

1.2 Physical cloning attacks

While the majority of research has focused on developing advanced machine learning (ML)-based models, some studies have concentrated on creating cloning attacks, particularly targeting weak PUFs. These attacks aim to replicate the unique characteristics of PUFs to produce duplicates that generate identical responses.

A variety of promising techniques, ranging from non-invasive to fully invasive, have been documented in the literature. These techniques exploit physical phenomena to effectively clone PUFs, with notable success observed in memory- and RO-based PUF designs.

In the context of RO-PUFs, [226] demonstrated that targeted ageing can be used to clone the behaviour and responses of RO-PUFs. Similarly, [227] utilized BTI-ageing techniques to target SRAM cells, successfully duplicating PUF responses. These attacks are particularly concerning because the circuitry used to evaluate PUFs is often left unsecured.

The studies [93, 92] employed high-resolution imaging to capture the unique power-up states of SRAM cells, replicating their physical structure to duplicate response characteristics. The authors highlighted the vulnerability of SRAM-PUF implementations, noting that memory-based PUFs offer little additional protection compared to industry-standard programmable NVM memories such as flash and EEPROM.

Several countermeasures to thwart cloning and side-channel attacks have been proposed in the literature. Notable examples include the works of [87] and [223]. However, side-channel attacks continue to be a critical issue in electronics, necessitating ongoing research and development of robust defence mechanisms.

Cloning attacks, leveraging both invasive and non-invasive techniques, present a substantial threat to the integrity of PUFs. Despite advancements in countermeasures, the persistent vulnerability to side-channel attacks underscores the need for continued innovation in securing PUF-based systems.

1.3 Mathematical and numerical modeling

Recent efforts in the literature have increasingly focused on building mathematical PUF models. These methods offer significant advantages, such as being non-invasive and requiring smaller datasets for training compared to machine learning (ML)-based methods. While these models can be exploited by attackers to replicate or predict PUF responses, they also serve a legitimate purpose in design verification akin to *stochastic models* in the context of True Random Number Generators, providing robust proof of the entropy source. Despite their critical importance, the field of mathematical modeling and verification of PUFs remains largely unexplored.

Established statistical models are utilized to accurately describe the probability distributions of PUF responses. Common modeling techniques include the use of linear models to approximate PUF responses and non-linear approaches to capture more complex PUF behaviours. Notable works in the literature, such as those in [228, 229] offer robust theoretical analysis and entropy estimation for delay-based PUFs. Similarly, [128] have contributed to modeling SRAM start-up behaviour for the refinement of SRAM-based PUFs.

One notable tool in this domain is PyPUF [230], which provides resources to create statistical models of strong PUFs for studying and predicting bit responses. However, attackers can leverage these models to predict the behaviour of specific PUF designs, as demonstrated in [231] for the Arbiter PUF.

Error Correcting Codes (ECCs) techniques used to ensure consistent and accurate PUF responses, could also greatly benefit from mathematical models that account for the impact of noise and operational variances. Robust physical characterization and modeling, such as those presented in [232], enhance the effectiveness of ECCs in PUF designs.

The mathematical modeling of PUFs is not limited to PUF designers; it is also performed from a cryptographic perspective to effectively generate secrets for secure protocols, as shown in [152]. This dual approach underscores the versatility and importance of mathematical models in both design and security contexts.

Less explored statistical frameworks, such as Bayesian Inference [233], could greatly benefit PUF analysis. Bayesian methods update the probability distribution of responses based on observed data. Indeed Bayesian and bootstrap techniques can be used to effectively extrapolate certain PUF metrics as showcased later. Additionally, the study of correlation effects could be enhanced using techniques like Fourier Analysis or Geospatial Analysis[176, 157, 162, 163].

Advanced statistical techniques are promising for PUF assessment. Frameworks like Surrogate Models [234, 235] aim to create accurate statistical models to predict responses from a minimal input subset, thereby speeding up the verification process. Another significant technique is Response Surface Methodology [236], which consists of mathematical and statistical methods to develop accurate models for predicting variables of interest based on a series of input parameters.

Despite the potential of these advanced tools, they remain underutilized across various scientific domains and need thorough examination before being integrated into PUF assessment methodologies. A comprehensive exploration and application of these statistical frameworks could significantly enhance the reliability and security of PUF technologies.

Modeling

VIII.2

As it has been presented in the previous section, the study of mathematical models for weak PUFs remains largely underexplored although promising work that provides entropy estimation methodologies as portrayed in [237, 173, 237, 170]. Consequently, this chapter addresses this gap by examining different suitable statistical and numerical methodologies. At first, certain mathematical considerations for various statistical distributions suitable for modeling PUFs are discussed. Following this, two initial models for RO-PUFs are proposed. These models leverage the relationship between frequency difference, Entropy, and Reliability, as demonstrated in this manuscript. However, it becomes evident that these models cannot be directly extrapolated to other PUF families due to the idiosyncrasies of each randomness source. This limitation, nonetheless, presents an opportunity for developing mathematical models capable of robustly predicting a wide range of PUF families.

2.1 Considerations for Distribution Selection

The normal distribution is ubiquitous across numerous scientific disciplines due to its detailed study and convenient mathematical properties, making it suitable for various types of analyses, such as linear regression and hypothesis testing. These properties extend to PUF evaluation metrics as well. However, there are compelling arguments for the use of the Beta distribution in the context of PUF evaluation.

Firstly, the Beta distribution is defined on the interval $[0, 1]$, which aligns well with the range of canonical PUF metrics and most normalized measurements. This bounded support makes it conceptually more appropriate for data constrained within this interval.

Secondly, while the Normal distribution is characterized by its bell shape, the Beta distribution is highly flexible, capable of assuming various shapes, including uniform, U-shaped, and bell-shaped. This flexibility allows for more accurate modeling of diverse PUF designs.

Lastly, the Beta distribution has a theoretical connection with Bit-aliasing. It is frequently used in inference analysis as the conjugate prior probability distribution for the Bernoulli, binomial, negative binomial, and geometric distributions. Given that each CRP can be represented by a unique binomial distribution, it is expected that the probabilities associated with CRPs follow a Beta distribution. As highlighted in previous sections, PUF assessment methodologies could significantly benefit from statistical techniques such as Bayesian inference.

This however, does not imply that the Normal distribution is unsuitable. Under the Central Limit Theorem, the Normal distribution can be a very good approximation, given sufficient

data. As demonstrated in Figure 56, both the Beta and Normal distributions yield almost identical results in our case.

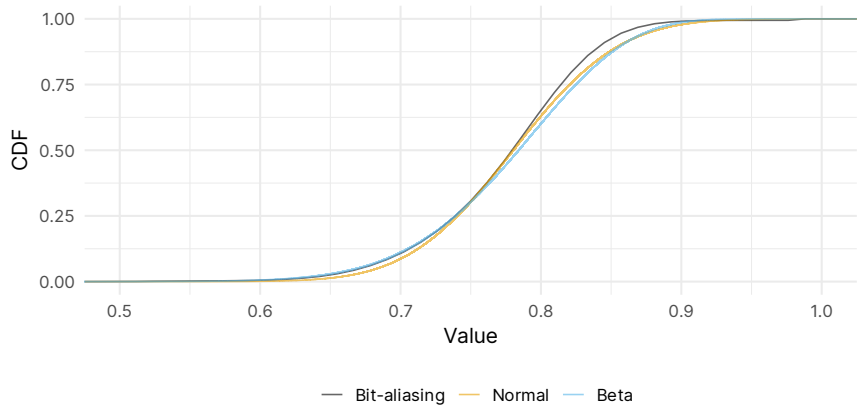


Figure 56: SRAM Bit-aliasing fitting of the Normal and Beta distributions

In conclusion, while the normal distribution remains a powerful tool due to its well-known properties and broad applicability, the Beta distribution offers specific advantages for PUF evaluation, particularly in terms of its bounded support and flexibility.

2.2 Proposed Reliability model

This section introduces an alternative method for Reliability modeling, as many others have already proposed [173]. Unlike the theoretical models, this approach is based on empirical results presented previously. The core observation is that for RO-PUFs, frequency differences close to zero are highly unreliable, whereas extreme frequency differences correspond to maximum reliability. The transition between these two effects is expected to be smooth, reflecting the Gaussian nature of the frequency differences. Among various mathematical functions that can model this behaviour, the logistic function is proposed here due to its common usage and ease of parameterization. The logistic function is generally parameterized as shown in Equation 2.1:

$$\text{Logistic} = f(x) = \frac{L}{1 + e^{-k(x-x_0)}} \tag{2.1}$$

where:

- L is the carrying capacity, the supremum of the values of the function.
- k is the logistic growth rate, the steepness of the curve.
- x_0 is the x value of the function's midpoint.

The standard logistic function, usually has parameters $L = 1$, $k = 1$, and $x_0 = 0$. An extended model for reliability, parameterized by $\beta_1, \beta_2, \beta_3, \beta_4$, is proposed in Equation 2.2:

$$\text{Reliability}(z_\Delta) \sim \beta_0 + \frac{1 - \beta_0}{\beta_1 + \exp[-\beta_2 \cdot (|Z_\Delta| - \beta_3)]} \tag{2.2}$$

where:

- β_0 controls the vertical offset from 0, representing the lowest possible reliability.
- β_1 controls the maximum reliability.
- β_2 determines the rate of reliability change (the slope of the curve).
- β_3 controls the horizontal offset from 0.

These parameters correspond to the logistic function's parameters as follows:

- β_0 corresponds to the minimum reliability at $z_\Delta = 0$, typically close to 0.
- β_1 is analogous to L, representing the maximum reliability for large values of z_Δ.
- β_2 is analogous to k, controlling the rate of change in reliability with respect to frequency differences.
- β_3 is analogous to x_0, determining the shift from $z_\Delta = 0$.

It is reasonable to assume no offset ($\beta_0 = 0$) in most situations, as a frequency difference of zero is inherently unreliable due to its uncertainty. The reliability curve is thus dependent on $|z_\Delta|$, with additional parameters potentially shifting the curve horizontally or altering its slope, which are presumably functions of z_Δ alone.

Future research should aim to incorporate confidence intervals or interval computation into this reliability model. This enhancement would provide a more comprehensive assessment of the model's reliability predictions.

2.3 Proposed Entropy model

Previous results demonstrate that the relationship between the entropy yielded by a RO pair and its frequency difference is inversely proportional: as the frequency difference increases, the resulting entropy decreases. Given the Normal distribution of frequency differences, the entropy can be modelled with a curve resembling a Gaussian distribution, albeit with slightly different behaviour. Specifically, maximum entropy is achieved when the frequency difference (f_Δ) is zero and decreases as the frequency difference increases.

To capture this behaviour, the following model for Entropy is proposed, parameterized by α_1 and α_2, as shown in Equation 2.3:

$$\text{Entropy}(z_\Delta) \sim \exp(-|\alpha_1 \cdot z_\Delta|^{\alpha_2}) \tag{2.3}$$

In this model:

- α_1 controls the width of the distribution.
- α_2 determines the "convexity" or "concavity" of the function, effectively the rate at which entropy decreases. When $\alpha_2 > 1$, the curve is concave, and when $\alpha_2 \in [0, 1]$, the curve is convex.

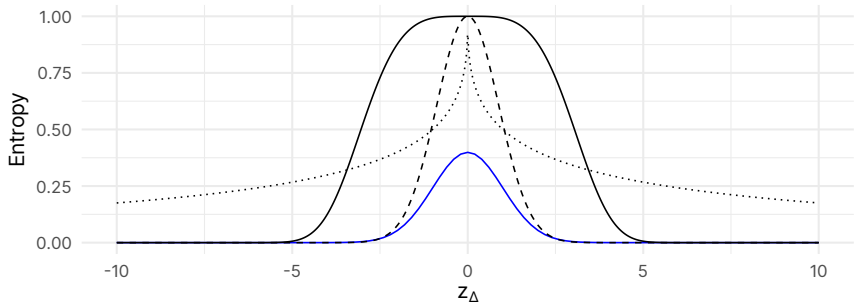


Figure 57: Entropy model for the RO-PUF

Analogous to the proposed Reliability model, further research is necessary to relate the parameters α_1 and α_2 to the distribution of frequency differences and the characteristics of the PUF designs. Understanding these relationships will enable more accurate model and prediction of entropy based on frequency differences in various PUF configurations.

2.4 Unification of the models

From this point onwards, z_Δ will be considered as positive. That is to say the absolute value is removed since the behaviour of the model is symmetric around 0. This will in turn simplify the algebra, by not needing to unfold the equations. With these modifications, the new resulting equations are shown in Equation 2.4

$$\text{Entropy}(z_\Delta) = \exp\left[-(\alpha_1 \cdot z_\Delta)^{\alpha_2}\right] \tag{2.4}$$

$$\text{Reliability}(z_\Delta) = \beta_0 + \frac{1 - \beta_0}{\beta_1 + \exp[-\beta_2 \cdot (Z_\Delta - \beta_3)]} \tag{2.5}$$

Initially the inverse function of Equation 2.4 is derived in order to compute Reliability as a function of Entropy.

$$\text{Entropy}^{-1}(\text{Entropy}) = \frac{1}{\alpha_1} \ln\left(\frac{1}{\text{Entropy}}\right)^{\frac{1}{\alpha_2}} \tag{2.6}$$

$$\text{Reliability}(\text{Entropy}) = \beta_0 - \frac{\beta_0 - 1}{\beta_1 + \exp\left[\beta_2\left(\beta_3 - \frac{1}{\alpha_1}\ln\left(\frac{1}{\text{Entropy}}\right)^{\frac{1}{\alpha_2}}\right)\right]} \tag{2.7}$$

Inversely, the inverse function of Equation 2.5 can be derived to compute the Entropy as a function of Reliability.

$$\text{Reliability}^{-1}(\text{Reliability}) = \frac{1}{\beta_2}\left[\beta_2\beta_3 + \ln\left(\frac{-\beta_0 + \text{Reliability}}{\beta_0\beta_1 - \beta_0 - \beta_1\text{Reliability} + 1}\right)\right] \tag{2.8}$$

$$\text{Entropy}(\text{Reliability}) = \exp\left[-\left(\frac{\alpha_1}{\beta_2}\left(\beta_2\beta_3 + \ln\left(\frac{-\beta_0 + \text{Reliability}}{\beta_0\beta_1 - \beta_0 - \beta_1\text{Reliability} + 1}\right)\right)\right)^{\alpha_2}\right] \tag{2.9}$$

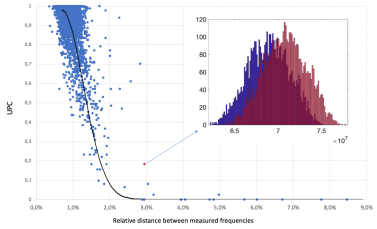
It's important to state that these models proposed may only work in real RO-PUFs implementations that don't exhibit bias. Further work would be needed to confirm how robust this model against different implementations. Equally important would be the automatic calculation of the parameters of both models given the dataset or certain characteristics of the design.

Furthermore, in order to extend this model to other PUF families and to account for more effects, interpolation techniques (i.e. Lagrange Interpolation, Newton's Divided Differences) along with other families of functions could be used to provide an assortment of function to choose from that better fit the design.

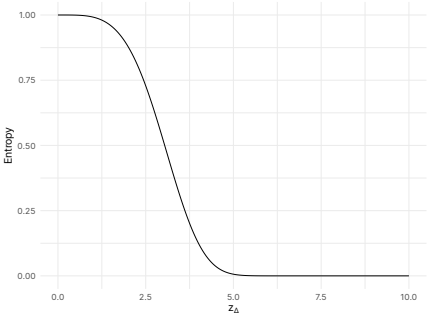
2.5 Evaluation on experimental data

To assess the accuracy, the proposed models will be evaluated on the Infineon Dataset. The set of parameters θ_α and θ_β have been fitted to approximate the results. A side by side comparison of the real data and the model is shown in Figure 58.

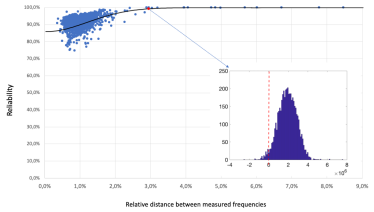
By using the relationship derived in Equation 2.9, the Entropy can be computed directly with the Reliability. As seen, certain numerical instabilities occur if the closed range $[0, 1]$ is used. To remove the instabilities, the open interval $(0, 1)$ should be used instead. While it would be ideal to refine the model to use the closed interval and remove the instabilities, a lot of statistical and inference techniques also suffer from the same problem.



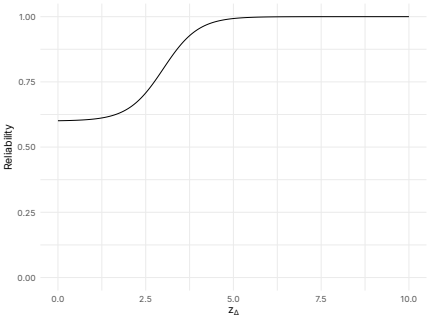
(a) Infineon RO Entropy



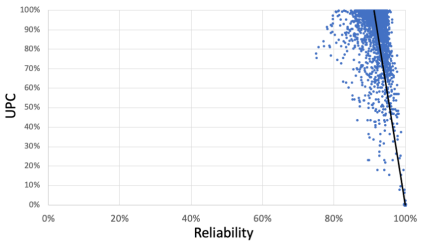
(b) Entropy model



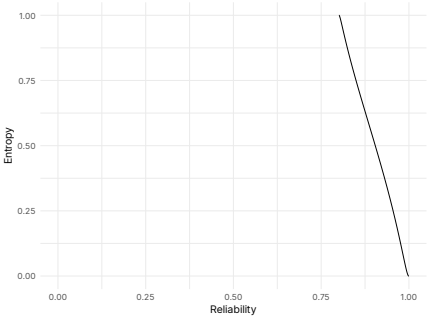
(c) Infineon RO Reliability



(d) Reliability model



(e) Infineon RO Reliability vs Entropy



(f) Reliability vs Entropy model

Figure 58: Evaluation of the model on the Infineon Dataset

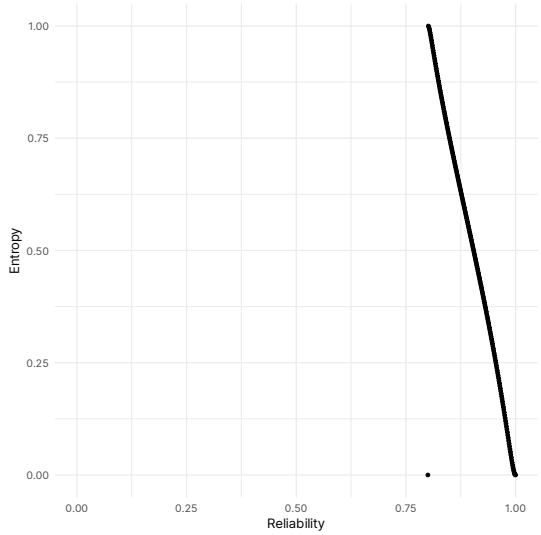


Figure 59: Entropy as function of Reliability using the derived expression

2.6 Response classification and labeling

The model proposed above can be extended to be used like a response classification methodology. PUF responses can be classified according to their performance across two different axes: The quality of the responses in space (e.g. Uniformity, Bit-aliasing and Uniqueness) and the quality in time (e.g. Reliability).

To provide a normalized classification, both axis will be normalized using Shannon Entropy, as depicted in (2.11).

$$\text{Entropy}(r) = \text{H}(\text{Bit-aliasing}) \tag{2.10}$$

$$\text{Stability}(r) = 1 - \text{H}\left(HW_F(r)\right) \tag{2.11}$$

Each PUF response could be plotted in the Cartesian graph $[0, 1] \times [0, 1]$. According to the graph, we can classify the responses according to the region they belong:

- Deterministic: Minimum Entropy (All devices have the same response) and maximum Stability (Same responses in time)
- PUF: Maximum Entropy (Different responses across devices) and maximum Stability (Same responses in time)
- TRNG: Maximum Entropy (Different responses across devices) and minimum Stability (Each device produces random values in time)

- PRNG: Minimum Entropy (All devices produce the same response, since the algorithm is deterministic and we assume all have the same seed) and minimum Stability (Each device produces random values in time. If they have the same seed obviously all responses will be the same across devices, but random in time)

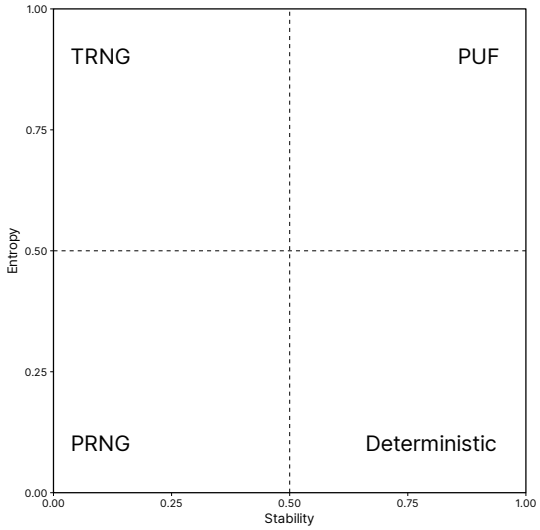


Figure 60: PUF response classification based on the relationship between Stability and Entropy

The responses from the SRAM PUF have also been classified and the results are shown in Figure 61. The heavy Bit-aliasing effects already shown are clear, since the majority of the bits are grouped in the TRNG quadrant. Only a few number of responses yield enough entropy to be valid for PUF use. However, almost no response is stable enough to be used reliable for identification.

Furthermore, this labelling methodology can be used to prove the claims that NBTI improves the reliability of the SRAM cells. If the hypothesis is correct, the points in the graph should be shifted to the right. As showcased in Figure 62 the effect is the opposite. This could be an indication that the reference used to create the data to write to the memory is not the good one. Since the Reliability of the SRAM-PUF is quite low, the reference sample obtained was the unreliable one. This is precisely the problem that was depicted in Chapter 2.

2.7 Conclusions

In conclusion, the proposed models for Reliability and Entropy accurately predict the behaviour of RO-PUF designs. Despite their simplicity, these models provide robust predictions of PUF responses. They could be utilized in ECCs or filtering strategies to enhance PUF performance in practical applications. Furthermore, the classification methodology based on these models effectively summarizes the overall behaviour of PUF responses and

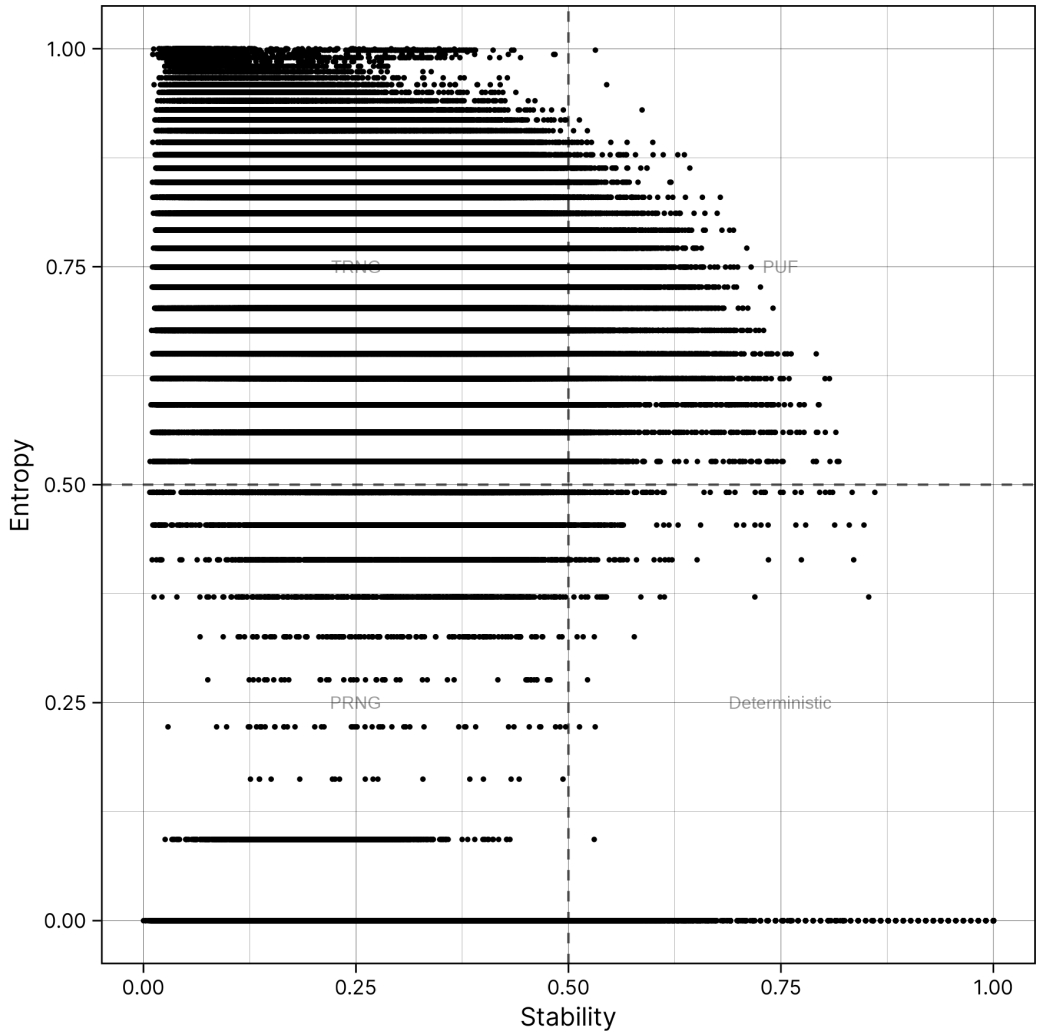


Figure 61: SRAM response classification.

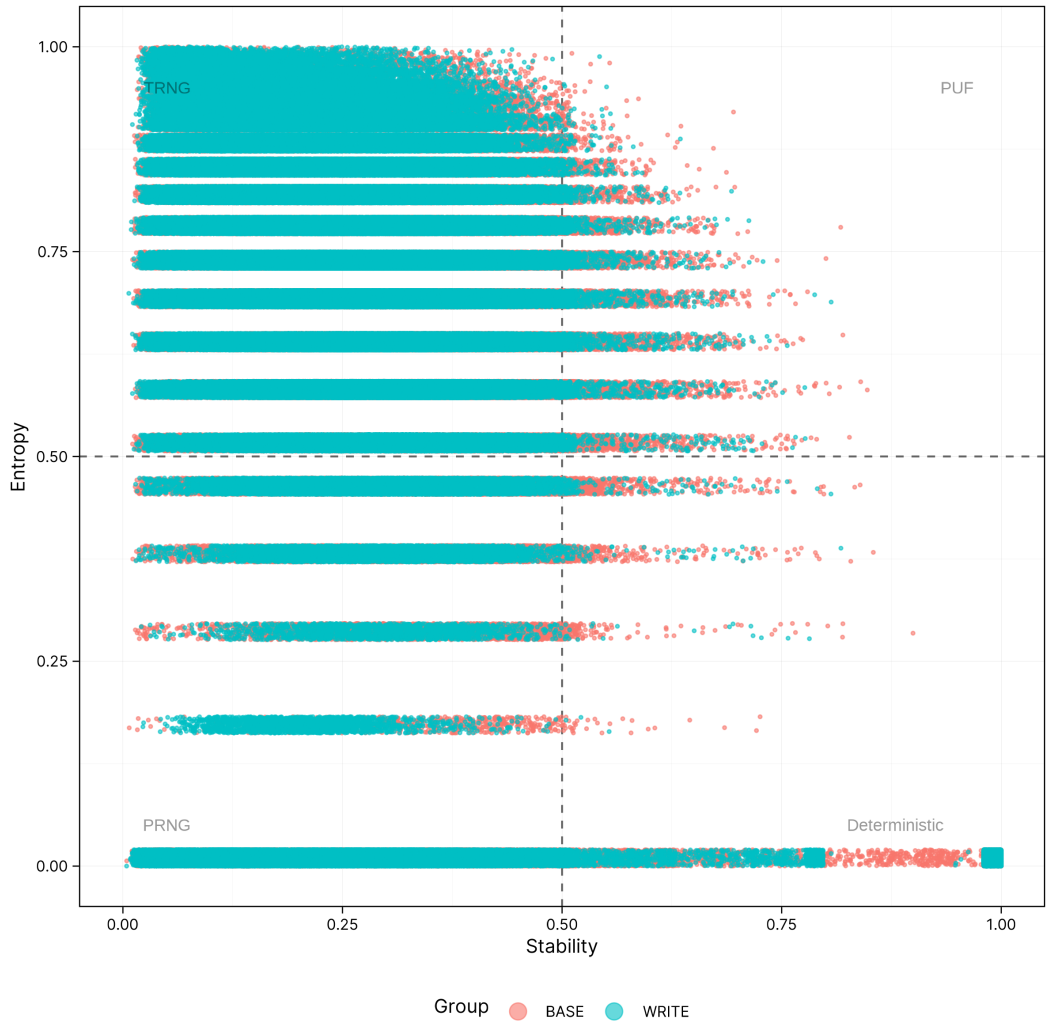


Figure 62: SRAM response classification for normal and NBTI test samples.

serves as a valuable tool for labelling different PUF responses. This labelling mechanism could be leveraged by unsupervised machine learning techniques for more robust filtering techniques.

Future work should focus on studying different PUF designs using the proposed models and classification methodology. The hypothesis is that the unique characteristics of each randomness source in various PUF designs will present distinct distributions of points in this graph. This classification could also be employed by unsupervised learning techniques, for example, in filtering strategies or to identify different randomness sources. Further research is necessary to confirm this theory. Additionally, incorporating other factors, such as correlation effects, could be crucial for the accurate classification of responses and represents another interesting avenue for future research.

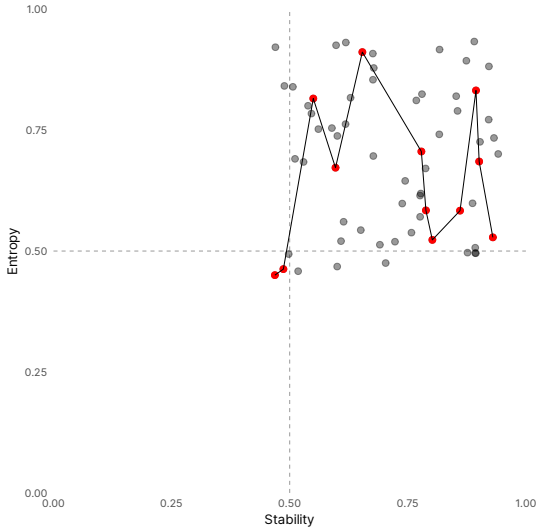


Figure 63: Proposal for PUF classification to account for correlation effects

A proposal to highlight the correlation between responses. Only responses that have a relationship higher than a defined threshold are considered correlated and the critical correlated pairs are connected. The correlation could be defined in one of many ways as presented in Chapter 2.

Relationship between PUF Metrics

VIII.3

As highlighted in the literature by[1][2], there is evidence suggesting that some canonical metrics are redundant. The study in this section is motivated by the work in[3], where the authors imply an interdependence and overlap among quality metrics. This analytical analysis is crucial as researchers typically compute metrics independently without understanding how the results may be related. For instance, Bit-aliasing may be computed, yielding poor results, and Uniqueness is computed regardless, despite the fact that the Uniqueness result depends on Bit-aliasing. Extending these studies and exploring the relationships between metrics would be highly beneficial, potentially reducing computation costs by allowing evaluation through a single set of sufficient statistics. This, in turn, would enable designs and analyses to focus on improving the sufficient statistic of the PUF, rather than undergoing multiple iteration cycles to assess improvements.

The analysis in this section aim to exploit certain approximations and analytical developments in order to describe a large part of the behaviour of the PUF only with Bit-aliasing. That is to say, if Uniformity and Uniqueness can be expressed in terms of Bit-aliasing, it should be considered a good enough sufficient statistic. This does not mean that Bit-aliasing is sufficient to evaluate the PUF performance, but the opposite.

$$\begin{aligned}\text{Uniformity} &= f(\text{Bit-aliasing}, C, D)\\ \text{Uniqueness} &= f(\text{Uniformity}, \text{Bit-aliasing}, C, D) \Rightarrow f(\text{Bit-aliasing}, C, D)\end{aligned}$$

While Bit-aliasing and Uniformity technically measure the Entropy albeit on different dimensions, the former is preferred to represent the state of the PUF for the following reasons:

- Loss of granularity, as all information about individual challenges is masked, that is to say we know the probability of 1 in each device. But we cannot know where the 1s are
- As highlighted in this document and in the literature, the sample set may not fully represent correctly the behaviour of the PUF deployed on the full system.

Furthermore, by integrating also the interrelation between Reliability and Entropy, the idea is to achieve a model that is able to represent the state of the PUF at any time given Bit-aliasing and Reliability

[1] L. Feiten *et al.*, 'On metrics to quantify the inter-device uniqueness of pufs,' *Cryptology ePrint Archive*, 2016.
[2] C. Gu *et al.*, 'A theoretical model to link uniqueness and min-entropy for puf evaluations,' *IEEE Transactions on Computers*, vol. 68, no. 2, pp. 287–293, 2018.
[3] F. K. Wilde, 'Metrics for physical unclonable functions,' Ph.D. dissertation, Universität München, 2021.

$$\text{PUF state at any time} = f(\text{Bit-aliasing}, \text{Reliability}, \text{Challenges}, \text{Devices})$$

The next challenge is to find the relationship between process variability and Bit-aliasing in order to have a robust framework to evaluate, optimize and discriminate specific PUF designs. However, that is outside of the scope of this thesis.

3.1 Analysis methodology

The analysis hase been tackled from 2 different approaches: (i) Grid Search to study synthetic data and (ii) Analytical development from the canonical metrics to finally strive for a middle-ground approach to verify the feasibility of certain approximations.

To ease the analytical development, the name of the metrics have been shortened in the equations: Uniformity is shortened to U or Unif, Bit-aliasing to BA and Uniqueness to $Uniq$, unless otherwise said.

The Grid Search algorithm has been implement in Julia and is shown in the Appendix is E.2. It is summarised in 2.

Data: $BA_\mu \subseteq \{0, \dots, 1\}$
Data: $BA_\sigma \subseteq \{0, \dots, 1\}$
Data: Devices $\subseteq \{2, \dots, 100\}$
Data: Challenges $\subseteq \{2, \dots, 1024\}$
foreach μ *in* BA_μ **do**
  foreach σ *in* BA_σ **do**
    $M \leftarrow \text{GenerateCRPs}(\mu, \sigma)$;
    foreach D *in Devices* **do**
      foreach C *in Challenges* **do**
        $\text{U} \leftarrow \text{Uniformity}(M_{\{1\to D, 1\to C\}})$;
        $\text{BA} \leftarrow \text{Bitaliasing}(M_{\{1\to D, 1\to C\}})$;
        $\text{Uniq} \leftarrow \text{Uniqueness}(M_{\{1\to D, 1\to C\}})$;
        $P_{diff} \leftarrow 2\text{BA}(1 - \text{BA})$;
        $P_{eq} \leftarrow \text{BA}^2 + (1 - \text{BA})^2$;
        $Cov_{BA} \leftarrow \text{Cov}\left[\text{BA}, \text{BA}^2\right]$;
        $Cov_{p_{diff}} \leftarrow \text{Cov}\left[P_{diff}, P_{diff}^2\right]$;
      end
    end
  end
end

Algorithm 2: Grid Search Pseudocode.

3.2 Bit-aliasing and Uniformity

The first Metric to derive from Bit-aliasing is Uniformity due to how tightly coupled they are. Following our representation of the PUF CRPs as a 2D matrix, Bit-aliasing is computed as the average value per column and Uniformity as the average value per row. Since the average value of each column is known, the average value per row will be the same as the average value per column, divided by the number of columns. This is equivalent to saying "The probability of finding a 1 in the whole PUF is the same whether we look by rows of by columns", so the expected value of Uniformity is the same as the one of Bit-aliasing.

$$\mathrm{E}\left[\hat{U}\right] = \frac{1}{D}\sum_{i=1}^{D}\left(\sum_{j=1}^{C} BA_j\right) = \frac{1}{D} \times D \times \mathrm{E}\left[BA\right] = \mathrm{E}\left[BA\right]$$

If we assume that the elements within each column are independent and identically distributed (i.i.d) then we can infer the distribution of average values per row. If the columns are i.i.d, then by properties of variance and the Central Limit Theorem, the variance of Uniformity is the variance of the average values by columns divided by the number of rows.

$$\mathrm{Var}\left[\hat{U}\right] \overset{?}{\approx} \frac{\mathrm{Var}\left[BA\right]}{D}$$

While this approximation would be true in an ideal scenario, there is a critical limitation with this approach as the metric evaluation is heavily dependent on the sample dataset, so it's not possible to compute the variance of Uniformity directly from Bit-aliasing.

This in turn means that Bit-aliasing is not a sufficient statistic, and we need both Bit-aliasing and Uniformity of the sample data set. Statistical techniques should be employed to extrapolate the metrics of a test set into our population, as will be presented later.

3.2.1 Extrema of Uniformity

Supposing that the Bit-aliasing evaluation of our PUF is representative enough, we can approximate the expected extreme values of Uniformity by considering the best and worst case scenarios:

Minimum Uniformity is expected when all bits are 0 except the ones with $p = 1$

$$\mathrm{Uniformity}_{Min} = \frac{|\{p = 1 \mid p \in \text{Bit-aliasing}\}|}{C} \tag{3.1}$$

Maximum Uniformity is expected when all bits are 1 except the ones with $p = 0$

$$\text{Uniformity}_{Max} = \frac{C - |\{p = 0 \mid p \in \text{Bit-aliasing}\}|}{C} \tag{3.2}$$

However, and as said before, this is an approximation as the computations are heavily dependent on the sample set. However, it helps to measure the "average properties of a random device" and discrepancies of these computations with our data set could be an indicator of lack of good device sampling.

3.3 Establishing compound probabilities

A series of compound probabilities are established to enhance the analysis. The basis of these compound probabilities lies in the independent analysis of each challenge as a binomial distribution. Recall also that the probability of collision between 2 devices has been already derived in Chapter 2.

Given a single challenge with probability of 1 p and D devices, we can estimate how many pairs of bits will be equal (00 or 11) as shown in Equation 3.3

$$N_{eq} = \binom{D}{2} p^2 + \binom{D}{2} (1-p)^2 = \binom{D}{2} \left(p^2 + (1-p)^2\right) \tag{3.3}$$

The number of equal pairs can be normalized to obtain the probability of equal pairs and it's complementary, the probability of different pairs. The probability that 2 responses are equal given a specific challenge is computed as shown in Equation 3.4

$$P_{eq} = (P_1 \cap P_1) \cup (\overline{P_1} \cap \overline{P_1}) = P(11) + P(00) = BA^2 + (1 - BA)^2 \tag{3.4}$$

While the probability that 2 responses are different given a specific challenge is derived as follows. Notice that the formula resembles the Logistic map.

$$P_{Diff} = (P_1 \cap \overline{P_1}) \cup (\overline{P_1} \cap P_1) = P(10) + P(01) = 2 \times BA \times (1 - BA)$$

Through analytical analysis and confirmed in Figure 64 its shown that $P_{Diff} = 1 - P_{Eq}$.

Since these 2 compound probabilities depend only on Bit-aliasing, we can derive that P_{Diff} will be bounded in the range $[0, 0.5]$ while P_{Eq} will be bounded in the range $[0.5, 1]$.

Due to the inherent similarities of these compound probabilities with the binomial distribution, it would be beneficial to model them using the Beta distribution. However, in this work the parameters of Bit-aliasing and the compound probabilities are not derived directly. Instead, analytical approximations are derived to compute the expected value and the variance.

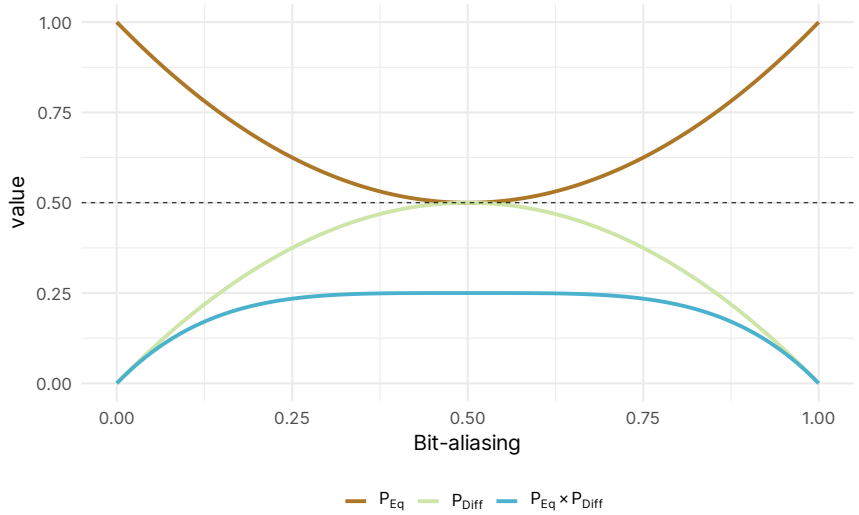


Figure 64: Relationship between P_{Eq} and P_{Diff}

3.4 Approximation of the compound probabilities

To approximate the expected value and the variance of the compound probabilities, the mathematical formulas are applied and the results are simplified by using the relationship between the expected value and the variance.

$$\mathrm{E}\left[P_{Diff}\right] = \mathrm{E}\left[2BA(1-BA)\right] \rightarrow 2(\mathrm{E}\left[BA\right] - \mathrm{Var}\left[BA\right] - \mathrm{E}\left[BA\right]^2) \tag{3.5}$$

$$\mathrm{Var}\left[P_{Diff}\right] = \mathrm{Var}\left[2BA(1-BA)\right] \Rightarrow 4 \times \left(\mathrm{Var}\left[BA\right] + \mathrm{Var}\left[BA^2\right] - 2\,\mathrm{Cov}\left[BA, BA^2\right]\right) \tag{3.6}$$

Given these approximations, it is evident now why the Grid search algorithm also computed $\mathrm{Cov}\left[BA, BA^2\right]$. While this may require its numerical computation since it's heavily dependent on the Bit-aliasing distribution, the Cauchy-Schwarz inequality can be used to determine some reasonable bounds.

Taking into consideration the Cauchy-Schwarz inequality $|\mathrm{Cov}(X,Y)| \leq \sqrt{\mathrm{Var}(X)} \cdot \sqrt{\mathrm{Var}(Y)}$ and the fact that $Var(BA) \in [0, a]$, where $a > 0$.

$$\mathrm{Cov}\left[X, X^2\right]_{Min} = -\sqrt{a} \times \sqrt{Var(X^2)}$$

$$\mathrm{Cov}\left[X, X^2\right]_{Max} = \sqrt{a} \times \sqrt{Var(X^2)}$$

For a random distribution of values bounded in the range $[0, 1]$, the largest possible variance corresponds to that of a Bernoulli distribution with equal probabilities. Recall that the variance of the Bernoulli distribution is computed as $pq = p(1-p) = 0.5^2 = 0.25$

In the worst case scenario then we obtain

$$-\sqrt{\frac{Var(X^2)}{4}} < \mathrm{Cov}\left[X, X^2\right] < \sqrt{\frac{Var(X^2)}{4}}$$

In the case of the Beta distribution, the highest variability should be the shape that put most of the values at the extreme of the distribution.

$$\mathrm{Var}\left[(\right]B) = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$$

Given the symmetry of the distribution, we set $\alpha = \beta = \theta$ to transform the formula of the variance of the beta distribution into the following:

$$\mathrm{Var}\left[B\right] = \frac{\theta^2}{4\theta^2(2\theta+1)} = \frac{1}{8\theta+4}$$

To maximize this, we need to minimize the denominator $8\theta + 4$. The smallest value of θ that is still positive is $\theta \to 0^+$. Thus, the variance approaches $\mathrm{Var}\left[X\right] \approx 1/4$.

For the Uniform distribution, the variance is defined as follows, where $b = 1, a = 0$.

$$\mathrm{Var}\left[X\right] = \frac{1}{12}(b-a)^2 = \frac{1}{12}$$

Furthermore, other mathematical functions can approximate the behaviour of P_{Diff} and P_{Eq}. The simplest ones are Shannon Entropy and the logistic map, as shown in Figure 65. Since the Shannon Entropy, more specifically, half of it, is representative of the uniqueness of the PUF, it shows that the Entropy is tightly coupled with the ability of the PUF to identify devices, which is at the foundation of the use of PUFs. However, this approximation fails for values of Bit-aliasing of approximately 0.25 and 0.75.

3.5 Deriving Uniqueness

Uniqueness can be approximated by using the compound probabilities established above. The following formulas have been derived from the data obtained with the Grid search and serve as approximations in most cases. However, further work will be needed to provide a generalized mathematical model that works in most situations.

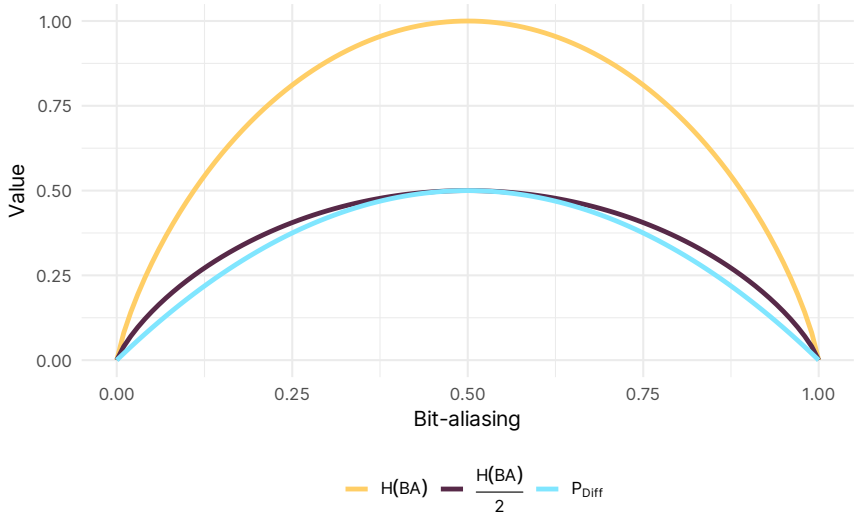


Figure 65: Comparison of different mapping functions.

$$\mathrm{E}\left[Uniq\right] = \frac{1}{C}\sum_{c=1}^{C} P_{Diff,c} \qquad \mathrm{Var}\left[Uniq\right] = \frac{\mathrm{Var}\left[P_{Diff}\right]}{D \times \sqrt{2}}$$

3.5.1 Extrema of Uniqueness

Analogous to Uniformity, an approximation to the extreme bounds of Uniqueness can also be performed by taking into account the best and worst case scenarios.

The minimum Uniqueness will be achieved when the number of bits with $P_{Diff} = 0.5$ is minimum

$$Uniqueness_{Min} = \frac{\left|\{p = 0.5 \mid p \in P_{Diff}\}\right|}{C} \tag{3.7}$$

While the maximum Uniqueness will be attained when the number of bits with $P_{Diff} = 0.5$ is maximum

$$Uniqueness_{Max} = \frac{\left|\{p = 0 \mid p \in P_{Diff}\}\right|}{C}$$

3.6 Conclusions and Extensions

As it's shown in the equations, we want $\mathrm{Var}\left[BA\right]$ to be as minimal as possible to improve the overall metrics, which translates to having a narrower distribution of Bit-aliasing. The

extreme case where there is no spread and every challenge has a probability of 1 of 0.5 occurs in a True Random Bit Generator.

The analysis described above assumes the distribution of Bit-aliasing is unimodal, which should be then norm in most situations. However, few works in the literature and the SRAM-PUF presented here presents a bimodal distribution of Bit-aliasing. While this is an isolated case, it is reasonable to think that this pattern may appear also in other PUF designs due to the symmetry effects of layouts or architecture. Because of this, we extend the analysis to the bimodal case. Note that while this will provide an extension to most of PUF designs (any other pattern outside of unimodal or bimodal is utterly broken) other analysis will need to be done, as most likely a PUF showing bimodality on the Bit-aliasing will likely be broken.

A series of approximations can be performed to apply the derived equations to each bell of the distribution. While these modifications may work for the dataset we have, they may not work for others. For this, further work will be needed to accurately describe the density function of P_{Diff} giving the distribution of Bit-aliasing. Moreover, mathematical analysis of mixture of Gaussians would be beneficial to automatically account for bimodal distributions.

Metric Verification Model

VIII.4

4.1 Introduction

As already said before, there is a lack of a reference model when it comes to evaluating PUFs under different system characteristics. Indeed, the same PUF designs needs to be evaluated under different number of devices and challenges in order to study the behaviour. Even then, there is no agreed consensus on the reference standard to compare against, because as depicted in the literature and highlight in this document, the PUF metrics change depending on the population size. In this section, an in-depth study of a golden PUF is presented, under varying system characteristics in order to derive a mathematical model or methodology, to relate the PUF and system characteristics to the reference values that shoul be attained.

4.2 Proposed methodology

The proposed methodology here is based on analysing the behaviour of an RO-PUF under different configurations. This analysis was tackled from 2 directions. The first one was purely numerical one, that is, perform a grid search on every possible scenario and evaluate the results to fit a model. The second one, is a purely analytical method and requires exploiting the relationships derived before. The idea was also to meet in the middle of both analysis to prove the analysis and to find some numerical or analytical optimizations.

The grid search algorithm is summarised in 3. Due to the sheer size of the search space the algorithm was implemented in Julia and it's source code is provided in **??**. Special attention is given to the study of Bit-aliasing, as it has shown before, we can relate the rest of the metrics with Bitaliasing. In order to include 0 in the set of mean frequency differences, the search size is set to 51 values as opposed to 50 values.

Even though the proposed methodology tries to obtain a reference model for any number of devices and challenges, the added computation effort of evaluating more than 500 devices and 512 challenges does not yield additional information.

4.3 Frequency Distribution and Bit-aliasing average

Recall the frequency difference distribution F_Δ in the context of RO-PUF represents the distribution of all computed frequency difference among RO pairs. Moreover, as depicted in previous sections, the expected value of Bit-aliasing is computed as the ratio of the areas of the distribution through 0, which can be computed easily with the CDF of the normal distribution as depicted in Equation 4.1 and shown in Figure 66.

Data: $\theta_\mu \subseteq \{-1\,\text{kHz}, \dots, 1\,\text{kHz}\}$ and $|\theta_\mu| = 51$
Data: $\theta_\sigma \subseteq \{0, \dots, 3\,\text{kHz}\}$ and $|\theta_\sigma| = 50$
Data: Devices $\subseteq \{2, \dots, 500\}$ and $|\text{Devices}| = 50$
Data: Challenges $\subseteq \{2, \dots, 512\}$ and $|\text{Challenges}| = 50$

```
foreach μ in θ_μ do
    foreach σ in θ_σ do
        F_Δ ← GenerateFreqDiff(μ, σ);
        M ← GenerateCRPs(F_Δ);
        foreach D in Devices do
            foreach C in Challenges do
                Study(μ, σ, M_{1→D,1→C});
            end
        end
    end
end
```

Algorithm 3: Grid Search Pseudocode.

$$\mathrm{E}\left[BA\right] = P(F_\Delta \geq 0) = 1 - P(F_\Delta < 0) = 1 - \Phi\left(Z_\Delta\right) \tag{4.1}$$

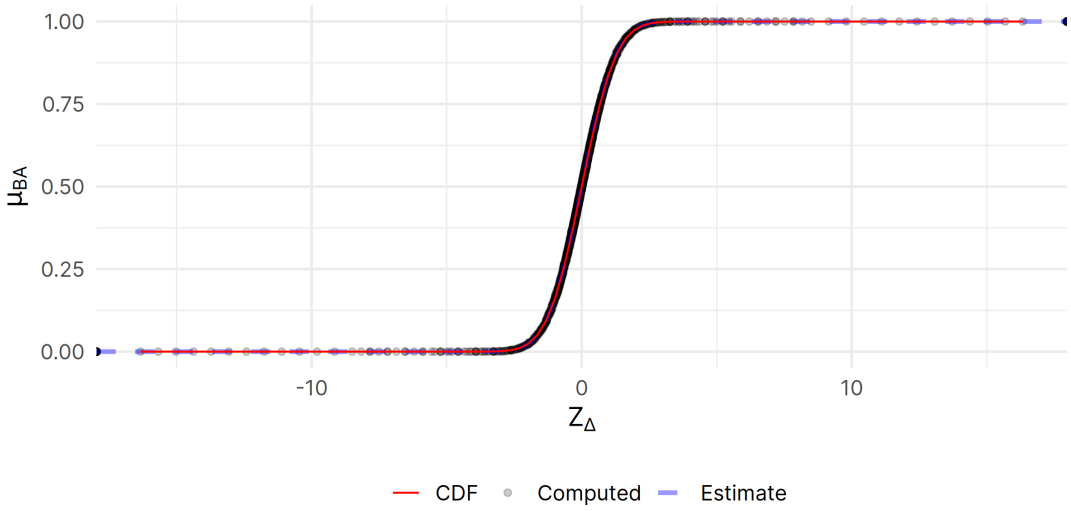


Figure 66: Relationship between Z_Δ and the expected Bit-aliasing

Also recall that the following expressions are equivalent since Z scores follow the standard normal distribution

$$\Phi\left(-Z_\Delta\right) = 1 - \Phi\left(Z_\Delta\right)$$

4.4 Variance of Bit-aliasing

The analysis of the variance of Bit-aliasing is non-trivial due to the complex relationship between C, D and Z_Δ

- Smaller Z_Δ implies larger σ_Δ and more chances of finding disparate f_Δ values that increase the variance of Bit-aliasing.
- The lower the number of challenges C, the more chances of finding disparate values of f_Δ that increase the variance of Bit-aliasing
- The higher the number of devices D, the more average the distribution of f_Δ is per column (i.e. challenge) decreasing the variance of Bit-aliasing.

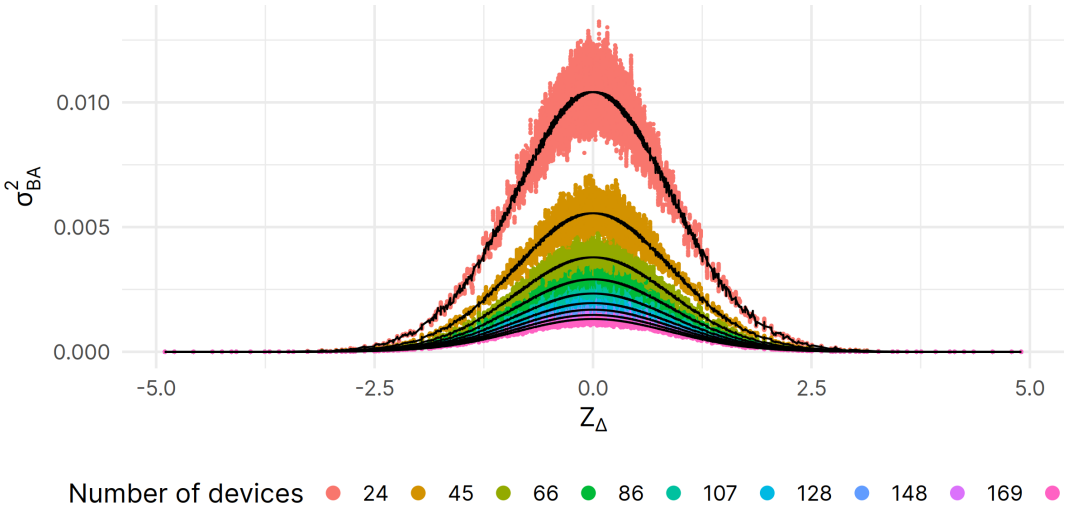


Figure 67: Bit-aliasing variance under different number of devices, challenges, and Z_Δ. The black lines represent the baseline estimation of Bit-aliasing Variance

The behaviour for a single number of devices is represented below in Figure 68. For very low values of C and D, the range is not exactly symmetrical with respect to the range. But for the sake of simplicity, we consider for now the case where the baseline passes exactly in the middle.

As it is clear, there is heavy heteroscedastic behaviour, so the model for the Bit-aliasing variance is proposed in Equation 4.2, where $\hat{S}$ is the baseline prediction for σ^2_{BA}.

$$\hat{\sigma}^2_{BA} \sim \beta_1 \hat{S} + \hat{\varepsilon} \tag{4.2}$$

The "uncertainty" $\hat{\varepsilon}$ is modelled as shown in Equation 4.3.

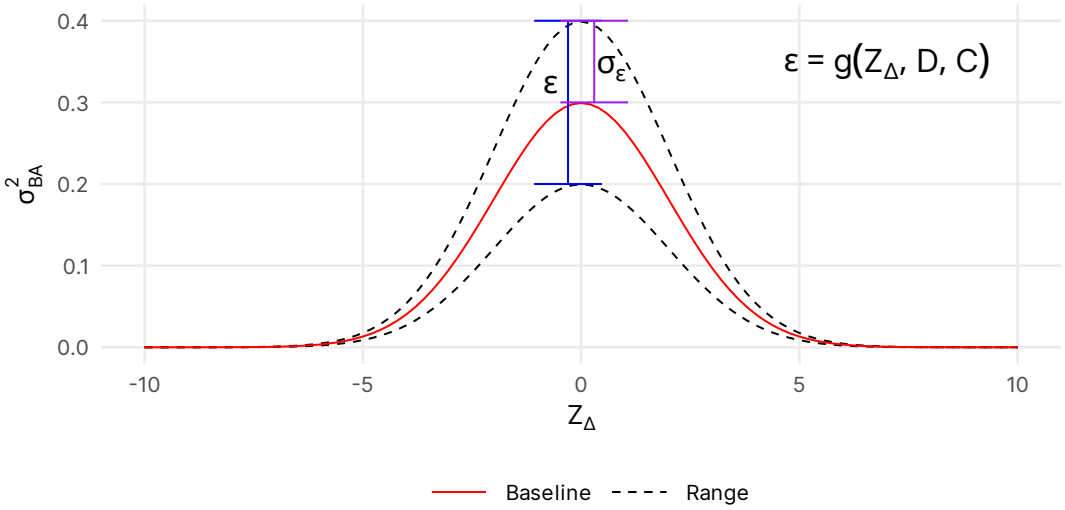


Figure 68: Relationship between Bit-aliasing variance and the baseline.

$$\hat{\varepsilon} \sim N(0, \sigma_\varepsilon) \qquad \sigma_\varepsilon = g(Z_\Delta, D, C) \tag{4.3}$$

The baseline $\hat{S}$ is computed as shown in Equation 4.4.

$$\hat{S} = \frac{\hat{\mu}_{BA}(1 - \hat{\mu}_{BA})}{D} \tag{4.4}$$

4.5 Analysis of residuals

In order to fit a model of the Bit-aliasing variance, we proceed to study the *residuals*, defined here as the difference between the expected value (i.e. the baseline) and the obtained value and they are normalized against their corresponding baseline.

$$R = \sum_{d \in D} \sum_{c \in C} \left[\sum_{z_\Delta \in Z_\Delta} \left| \text{Var}\left[BA\right]_{d,c,z} - \hat{S}_{d,c,z} \right| \right]$$

$$R' = \sum_{d \in D} \sum_{c \in C} \left[\sum_{z_\Delta \in Z_\Delta} \frac{\left| \text{Var}\left[BA\right]_{d,c,z} - \hat{S}_{d,c,z} \right|}{\hat{S}_{d,c,z}} \right]$$

However, due to the large amount of data, certain reductions are needed to simplify the computations and accurate fit the distribution. All the z-scores in the range $Z_\Delta \in [-5, 5]$ are split into 150 bins according to the following series of computations and the results are displayed in Figure 69.

$$Bin(z_\Delta) = \left\lceil \frac{z_\Delta + \max(Z_\Delta)}{Bin_{Width}} \right\rceil \qquad Bin_{width} = \frac{\max(Z_\Delta) - \min(Z_\Delta)}{N}$$

$$R_{env} = \max_{i=1}^{N} \{R : Bin(Z_\Delta) = i\}$$

$$Z_{env} = \{\overline{Z_\Delta} : Bin(Z_\Delta) = i \mid \forall i \in [1, N]\}$$

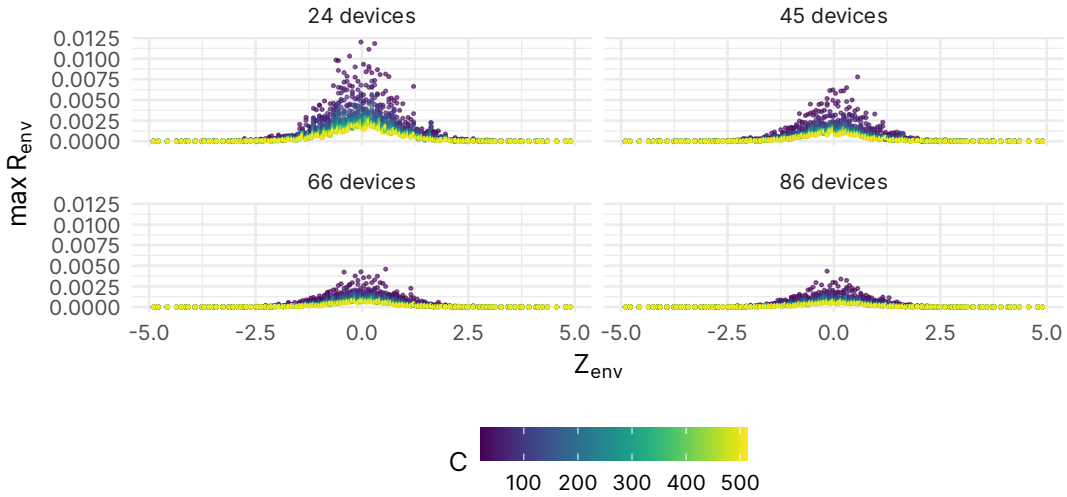


Figure 69: Relationship between number of devices, challenges and binnarized residuals

The R code shown in 4.1 was used to binarize the residuals.

```
cases_df <- cases_df |>
  filter(abs(z) <= 5, C > 4, D > 4) |>
  mutate(
    ba_mean_est = pnorm(z),
    ba_var_baseline = ba_mean_est * (1 - ba_mean_est) / D
  ) |>
  mutate(ba_var_residual = ba_var - ba_var_baseline) |>
  mutate(residual_norm = ba_var_residual / ba_var_baseline)

num_bins <- 150

df_binned <- cases_df |>
  mutate(z_bin = cut(z, num_bins)) |>
  group_by(z_bin, C, D) |>
  summarise(
    ba_var = max(ba_var, na.rm = TRUE),
    ba_var_baseline = mean(ba_var_baseline),
    max_residual = max(abs(ba_var_residual), na.rm = TRUE),
    z = mean(z, na.rm = TRUE)
  ) |>
  ungroup()
```

Listing 4.1: R code to binarize the results

The distribution that best fits the residuals is the Laplace distribution. In this case we can always assume that μ is 0 as the distribution of residuals is always centred at 0, so we only need to obtain the b parameter. The fitting is performed in R by using the code showed in 4.2.

```r
# Laplace Density Function
dlaplace <- function(x, mu, b) {
  (1 / (2 * b)) * exp(-(abs(x - mu)) / b)
}

# ba_var_residual ~ beta * laplace(x; mu=0, b)
laplace_fit <- function(df_group) {
  C <- unique(df_group$C)
  D <- unique(df_group$D)

  # Params are beta, b
  params <- c(exp(-C), 0.2)
  fun <- function(par, data) {
    v_est <- par[1] * dlaplace(data$z, 0, par[2])
    sum((v_est - data$max_residual)^2)
  }
  params <- optim(params, fun, data = df_group)$par
  list(D = D, C = C, beta = params[1], b = params[2])
}


# Parameters to optimise are are beta, b
laplace_fit <- function(df_group) {
  C <- unique(df_group$C)
  params <- c(exp(-C), 0.2) # Initial values
  fun <- function(par, data) {
    v_est <- par[1] * dlaplace(data$z, 0, par[2])
    sum((v_est - data$max_residual)^2 )
  }
  params <- optim(params, fun, data=df_group)$par
  list(D = D, C = C, beta = params[1], b = params[2])
}
```

Listing 4.2: R code to fit the residuals to the Laplace distribution

The distribution of residuals is dictated mainly by the b parameter of the Laplace distribution, whose behaviour can be modelled with Equation 4.5. While this expression describes the relationship between b, the number of devices and challenges, we can approximate b to 0.81.

$$b \sim 5.828 \times 10^{-5} D + 8.631 \times 10^{-5} C - 2.867 \times 10^{-7} (D \cdot C) + 0.7017 \approx 0.81 \tag{4.5}$$

Now that we have derived the expression for b, we can fit the data to obtain the model of σ_ε as the following.

$$\sigma_\varepsilon \sim \beta \cdot L(z_\Delta \mid \mu = 0, b \approx 0.81)$$

By fitting a non-linear model we obtain the model for σ_ε shown in Equation 4.6.

$$\beta \sim \frac{0.06145}{D} + \frac{11.49}{D \cdot C} - 1.30 \times 10^{-5} \tag{4.6}$$

4.6 Analysis of heteroscedasticity

To model the behaviour of the Bit-aliasing variance, an error term is added as follows

$$\mathrm{Var}\left[BA\right] = \mathrm{E}\left[\hat{BA}\right] + \hat{\epsilon} = \frac{\mathrm{E}\left[BA\right] * (1 - \mathrm{E}\left[BA\right])}{D} + \hat{\epsilon}$$

where the error term is normally distribution according to the following:

$$\hat{\epsilon} \sim N(0, \sigma_{Noise})$$

The standard deviation of the error term, σ_{Noise} is fitted as

$$\sigma_{Noise} = 0.18 \exp\left(\frac{1}{D^2}\right) + 0.92\frac{1/D}{1/C} - 0.18$$

4.7 Summary

The expected value of Bit-aliasing is computed directly given the Z_Δ of the distribution using the CDF of the normal distribution.

$$\hat{\mu}_{BA} = \Phi\left(Z_\Delta\right)$$

The variance of Bit-aliasing presents heteroscedasticity so the model is represented by a baseline $\hat{S}$ and the noise ε which is normally distributed.

$$\hat{\sigma}^2_{BA} = \hat{S} + \hat{\varepsilon} = \frac{\hat{\mu}_{BA}(1 - \hat{\mu}_{BA})}{D} + N(0, \sigma_\varepsilon)$$

The variance of Bit-aliasing is distributed following a Laplace distribution. The 1/3 scaling factor is added since the model accounts that the behaiour is representative of 3σ.

$$\sigma_\varepsilon \sim \beta \cdot L(z_\Delta \mid \mu = 0, b \approx 0.81) \cdot \frac{1}{3}$$

After performing a non-linear model, the scaling factor β of the Bit-aliasing variance is given by:

$$\beta \sim \frac{0.06145}{D} + \frac{11.49}{D \cdot C} - 1.30 \times 10^{-5}$$

4.8 Verification of the model

The model has been verified with synthetic data generated outside the search space used to develop the model. Figure 70 shows the graph of Bit-aliasing variance with respect to the Z_Δ of the frequency difference distribution. Each dot corresponds to a value. The red line corresponds to the baseline variance. The green dots have been generated using the models proposed above. As it's clear, most of the generated points fall within the range of the ideal values.

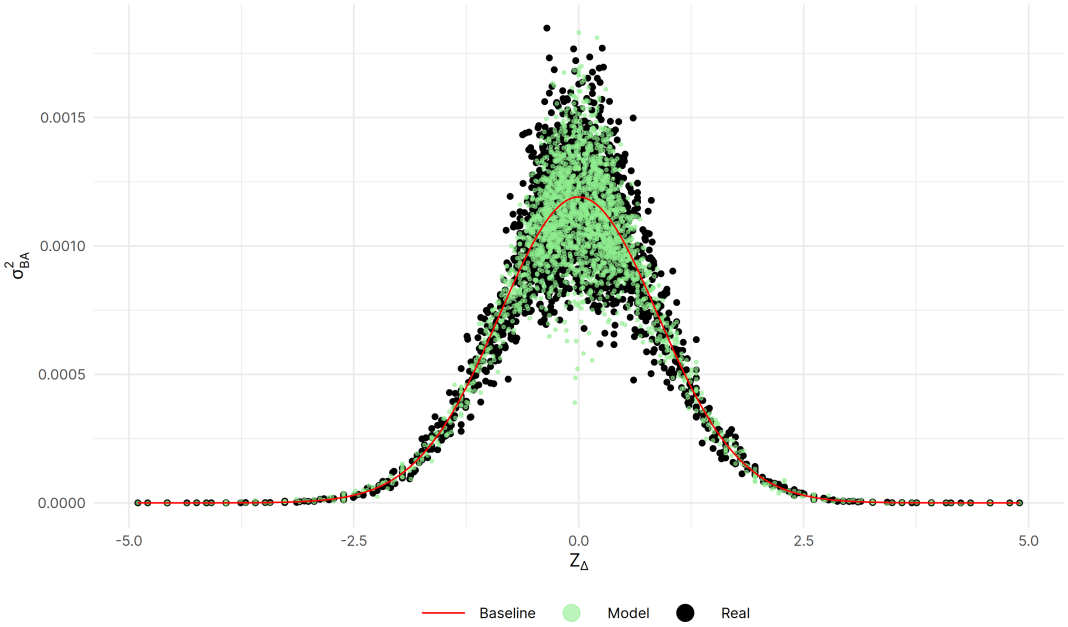


Figure 70: Evaluation of the model for 120 devices and 74 challenges

4.9 Conclusions

The model presented here serves as a robust reference for the behaviour of a RO-PUF and can be adapted to the intricacies of different PUF families.

To accurately represent the behaviour of the model, confidence intervals should be computed. The type of confidence interval to compute depends on the size of the system

under study, with the t-statistic being more appropriate for a low number of devices and challenges. For an accurate representation, Wild bootstrap[1], a variation of the standard bootstrap suited for models with heteroscedasticity, is proposed. However, the computation of the confidence intervals was not presented here due to time limitations.

An important addition would be to provide an explanation of the different parameters and factors in the models. Furthermore, studying PUFs that present biasing would be interesting to understand how deviations from ideal metric values translate into the model. This could allow for a "universal" way to compare PUF designs against a robust golden standard.

Given that this model can accurately represent standard behaviour for a given system configuration, it would be beneficial to perform sensitivity analysis. The goal is to study the effect of different system parameters on the metrics. This information can guide design efforts toward parameters that are "easily" changeable yet yield significant improvements. For example, increasing the number of RO stages to achieve a larger frequency deviation is much easier than creating designs that force the frequency difference distribution to be extremely close to zero.

[1] C.-F. J. Wu, 'Jackknife, bootstrap and other resampling methods in regression analysis,' *the Annals of Statistics*, vol. 14, no. 4, pp. 1261–1295, 1986.

Extrapolating PUF metrics

VIII.5

5.1 Introduction

The previous sections have provided an in-depth study about the interrelation between the different metrics and how Bit-aliasing can serve as a sufficient statistic to describe a PUF data set. However, certain statistical limitations appear, specially when computing Uniformity due to the loss of information about specific devices. This section expands on the models and mathematical analysis presented.

Since this models allow us to verify and compute some PUF metrics given the results of other metrics, the idea is to extrapolate the full set of metrics for a given system description (i.e. number of devices, number of challenges) given a small testing subset of devices. This has the potential to greatly reduce the time and cost needed to assess the performance of a PUF, as only a small sample set is needed, removing the needed to manufacture more devices if the expected behaviour is not seen in the test sample.

The methodology proposed here relies on parametric bootstrap in order to estimate the desired metrics from a small sample test whose metrics are known. It's important to state that this methodology relies only on the evaluation on a limited number of devices, so the number of challenges tested remains the same. Removing challenges may hide information and lead to incorrect results.

Due to the statistical nature of the bootstrap algorithm, a certain set of considerations need to be taken into account for accurate results. This algorithm is very dependent on the sample population. If the original sample is biased or unrepresentative, the bootstrap samples will also be biased, leading to wrong estimations, reiterating again the importance of a representative sample population. For very large populations, the original sample size should be a small fraction of the population to avoid overfitting. For smaller populations, the sample size should be large enough to adequately represent the population diversity. As bootstrap requires resampling of the population, the results depend also on the number of iterations performed. Common practice is to use at least 1,000 to 10,000 bootstrap samples. This helps in reducing the variability of the bootstrap estimates.

5.2 Proposed methodology

As described above, the methodology exploits the relationship between metrics in order to extrapolate known metrics from a small sample set into the desired population size. The full algorithm is summarized in 4.

Data: $M \leftarrow$ Matrix of CRPs
Data: $D \leftarrow$ Number of devices
Data: $\eta \leftarrow$ Ratio of sample size to population size
Data: Iters $\leftarrow$ Number of bootstrap iterations
foreach $i \leftarrow 0$ *in Iters* **do**
  $M' \leftarrow$ create_sample(M, D);
  $U_i^* \leftarrow$ Uniformity(M');
  $BA_i^* \leftarrow$ Bitaliasing(M');
  $BA_i^{2*} \leftarrow BA_i^* \cdot BA_i^*$;
  $P_{Diff,i}^* \leftarrow 2 \cdot BA_i^* \cdot (1 - BA_i^*)$;
end
$\theta_\mu^* \leftarrow$ compute_mean$(U^*, BA^*, BA^{2*}, P_{Diff}^*)$;
$\theta_{\sigma^2}^* \leftarrow$ compute_var$(U^*, BA^*, BA^{2*}, P_{Diff}^*)$;
Result: $\theta_\mu \leftarrow$ Extrapolate(θ_μ^*, η)
Result: $\theta_{\sigma^2} \leftarrow$ Extrapolate(θ_σ^*, η)

Algorithm 4: Bootstrap algorithm to extrapolate metrics.

Through this section, we will denote $\hat{X}$ as the extrapolated value of the metric X. Initially, we first define η as the ratio between the number of devices in our sample set against the desired number of devices in the population.

$$R = \frac{\text{Number of devices in the sample}}{\text{Number of devices in the population}}$$

As the expected value of Bit-aliasing depends only on the number of challenges, and that is fixed, we can say that the expected value in the population is practically identical to the one in our sample.

$$\mathrm{E}\left[\hat{BA}\right] \simeq \mathrm{E}\left[BA\right] \iff \text{Proper sampling}$$

However, the variance depends on the population size, so it can be extrapolated by taking into account the ratio η as follows

$$\mathrm{Var}\left[\hat{BA}\right] = \mathrm{Var}\left[BA\right] \cdot \eta$$

The expected value and variance of Uniformity are extrapolated directly from the bootstrap samples. Notice that the expected value of Uniformity is the same as the expected value of Bit-aliasing.

$$\mathrm{E}\left[\hat{U}\right] = \theta_\mu^* = \mathrm{E}\left[\hat{BA}\right] \qquad \mathrm{Var}\left[\hat{U}\right] = \theta_{\sigma^2}^*$$

The compound probability P_{Diff} can be extrapolated from the Bit-aliasing by using the relationship derived before in Equation 3.5 and Equation 3.6.

$$\mathrm{E}\left[\hat{P_{Diff}}\right] = 2\left(\mathrm{E}\left[\hat{BA}\right] - \mathrm{Var}\left[\hat{BA}\right] - \mathrm{E}\left[\hat{BA}\right]^2\right)$$

The variance of the P_{Diff} can be approximated by using the already derived relationship with Bit-aliasing. Due to the complex relationship between the variables, the absolute value is added to avoid negative values that may occur in certain occasions.

$$\mathrm{Var}\left[\hat{P_{Diff}}\right] = \left|4(\mathrm{Var}\left[BA\right] + \mathrm{Var}\left[BA^2\right] - 2\,\mathrm{Cov}\left[BA, BA^2\right])\right|$$

However, that approximation only provides reliable results for small population sizes. The following approximation yields much better results once the population grows large, in our case more than 200 devices. Future work will be needed to provide a generalized solution.

$$\mathrm{Var}\left[\hat{P_{Diff}}\right] = \frac{\eta^2}{4}\theta^*_{\sigma^2}$$

Finally, the Uniqueness is extrapolated as follows

$$\mathrm{E}\left[\hat{Uniq}\right] = \mathrm{E}\left[\hat{P_{Diff}}\right] \qquad \mathrm{Var}\left[\hat{Uniq}\right] = \frac{\eta}{4}\mathrm{Var}\left[\hat{BA}\right]$$

The confidence intervals of the metrics can be computed directly through the Pivot Confidence Interval if bootstrap was performed. Otherwise, the t-statistic is preferred here to the z-statistic due to the limited size of our sample.

While technically speaking, it's possible to derive all metrics numerically directly through bootstrap from the sample set, the already derived relationships are used to reduce the computation and time required to obtain the results. Nonetheless, in certain situations there are discrepancies between the estimated parameters and the results obtained through bootstrap, so it's advisable to check for any differences between them.

5.3 Evaluation of the methodology

To validate the methods proposed, a series of synthetic datasets were evaluated. These synthetic datasets provide full control on the system characteristics and allows the study of certain edge cases.

A series of studies was performed by changing the following parameters:

- The population size was test from 40 to 500 devices
- The sample to population ratio η was studied from 0.1 to 0.9

Metric	μ		σ^2	
	Full	Model	Full	Model
Uniformity	0.4997	0.5010	2.3538×10^{-5}	2.3882×10^{-5}
Bit-aliasing	0.4997	0.5010	3.0313×10^{-3}	3.0318×10^{-3}
P_{Diff}	0.4939	0.4939	7.3124×10^{-5}	7.5795×10^{-5}
Uniqueness	0.5002	0.4939	2.5705×10^{-5}	2.6797×10^{-5}

- The number of bootstrap iterations was changed from 1000 to 5000
- The bootstrap iteration size was fixed to the sample size.

To ease up the testing procedure, a Python module and a crude TK interface and Python module have been created. The Graphical User Interface (GUI) is shown in Figure 90. The full set of metrics is validated on the whole test set. The size of the test set, the number of bootstrap iterations and the number of samples per iterations are configured through the interface and the metrics are extrapolated from there. Both results are displayed to the user and compared.

While there are some minor discrepancies it's evident that most of the extrapolated values are very close to the computed ones.

5.4 Conclusions

The methodology proposed is able to extrapolate metrics from a small sample set with great accuracy. Due to the statistical nature of the bootstrap algorithm, the accuracy of the extrapolation depends on the population size and the number of iterations. We have shown here than at a ratio of $\eta \geq 30\%$ in needed. This method has huge potential in reducing the cost of PUF assessment and evaluation as only a few devices will be needed to assess the performance of the PUF when deployed in the full system.

Further work will be needed to achieve better accuracy and to include the automatic computation of the Confidence Intervals.

While this methodology provides a general view of the PUF behaviour, no correlation effects have been included in the analysis. Indeed, the addition of correlation and topological effects in the analysis will enhance this procedure. These effects could be accounted with the addition of others well established statistical techniques like the Metropolis-Hastings algorithm and the use of Markov Chain Monte Carlo (MCMC).

Another important remark is that this method works well with unimodal distributions. As already highlighted, we are not the only ones to detect bimodal patterns in a PUF dataset. So it's feasible to think that certain designs can produce this bimodal distribution. Further

work will be needed to account for bimodal and even mixture of distributions in order for this method to accurately work on every PUF dataset possible.

PUF Digital Twins

VIII.6

6.1 Introduction

As previously discussed, the mathematical modeling of PUFs remains largely unexplored. In fields where accurate statistical models are needed to reduce computational complexity, surrogate models have proven useful [235]. Other effective mathematical methods, such as those based on response surface methodology, have been successfully applied to optimize flexible printed circuit boards [236, 239]. However, these powerful mathematical abstractions are often complex and require extensive knowledge to implement.

The methodologies presented here propose the creation of PUF "digital twins" to reduce the cost and time required for evaluating real PUFs. A digital twin[1] is a digital representation of a real-world physical process that serves as an effectively indistinguishable counterpart for purposes such as simulation, integration, testing, monitoring, and maintenance.

The main distinction between the proposed methods and the ML-based approaches is tunability. The proposed methods include adjustable parameters, or "knobs," that can modify the model's behaviour. These "knobs," combined with pre- and post-processing stages, enable not only the simulation of a PUF but also its evaluation under different conditions and scenarios. Furthermore, the models are simple enough to be computationally very efficient while accurate enough to describe the behaviour of the PUF. This flexibility provides a comprehensive tool for understanding and optimizing PUF performance.

6.2 Proposed methodology

Simulating the behaviour of the PUF under its two main dimensions, spatial (i.e., entropy and correlation) and temporal (i.e., reliability and ageing), is essential. Each type of PUF may require a different methodology for cloning. This work particularly focuses on modeling the SRAM PUF due to its extensive study and the relative simplicity of bistable systems.

The simplest model to represent a challenge-response pair (CRP) is a binomial distribution, represented by a two-state Markov Chain.

However, this approach is too simplistic, failing to capture the spatial and temporal behaviour of the PUF. One limitation of Markov Chains is their "memory-less" property, where future states depend only on the current state and not on past states. In contrast, the current model aims to account for "short-term memory."

[1] S. Haag and R. Anderl, 'Digital twin–proof of concept,' *Manufacturing letters*, vol. 15, pp. 64–66, 2018.

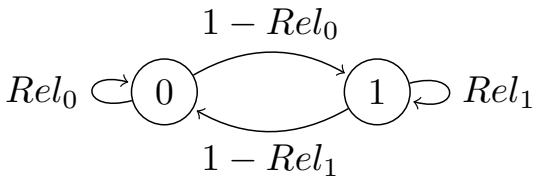


Figure 71: Binomial representation of a CRP using a Markov Chain

This short-term memory is closely related to the concept of Reliability Invariance proposed in this work. Some challenges are likely to remain in a specific state (e.g., reliable 0) but can still produce an unexpected bit (e.g., a response of 1).

To model this behaviour, Hidden Markov Chains (HMCs) are employed, as illustrated in Figure 72.

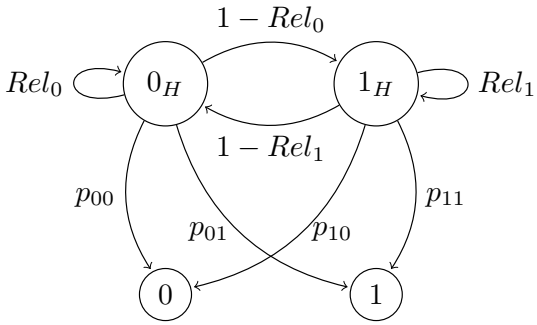


Figure 72: Representation of a CRP using a Hidden Markov Chain

In the HMC model, each response is initially modelled with two "hidden" or latent states. Each state has emission probabilities corresponding to the likelihood of yielding a 0 or 1, given the current state of the response, For each response, the distribution of Reliability Invariance (P00, P01, P10, P11) and the canonical reliability are computed per bit. To generate a digital device, values are drawn from each distribution and assigned to the states. The R code shown in 6.1 demonstrates how a Hidden Markov Chain is created for a single response. Note that all probabilities need to be normalized so they sum to 1.

```
make_hidden_cell <- function(ba, rel_0, rel_1, P00, P01, P10, P11) {
  norm <- function(v) v / sum(v)

  m <- list(c(rel_0, 1 - rel_0), c(1 - rel_1, rel_1))
  # m <- list(c(P00, P01), c(P10, P11))
  m <- lapply(m, norm)

  emmision <- list(c(P00, P01), c(P10, P11))
  emmision <- lapply(emmision, norm)

  state <- ifelse(ba ⩾ 0.5, 1, 0)
  structure(
    list(P = m, E = emmision, state = state, prev = state),
    class = "cell"
```

```
    )
}
```

Listing 6.1: Hidden Markov Chain implementation in R

The following graphs show the histogram of Uniformity and Bit-aliasing of the real SRAM CRPs and the generated ones using the Hidden MC model.

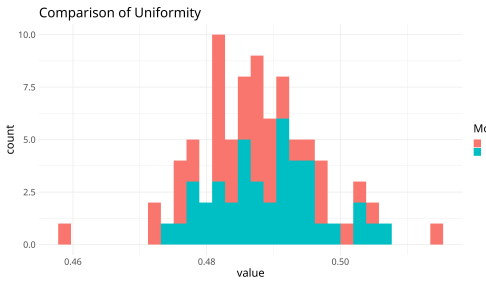


Figure 73: Uniformity Histogram

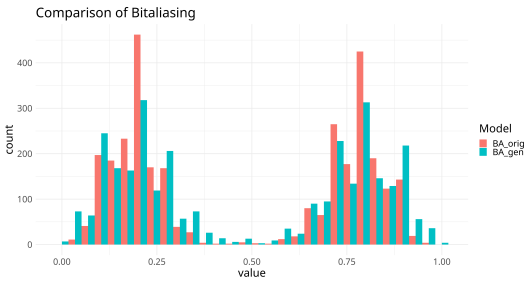


Figure 74: Bit-aliasing Histogram

The results show that the model tends to create the same distribution of responses when it comes to the spatial behaviour. This is also reassured by Figure 75 where the real CRPs of the SRAM are compared against to the generated ones. It's evident that the same pattern arises in the model, which is a good indication.

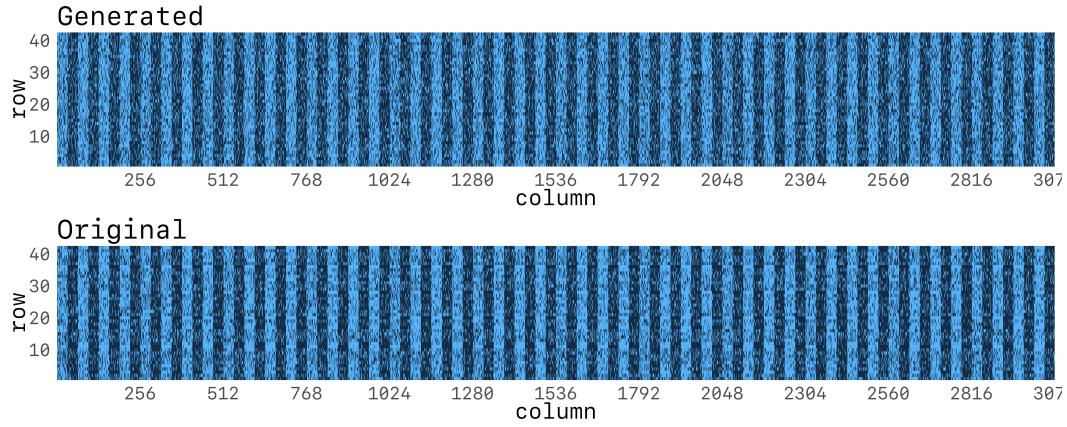


Figure 75: Original SRAM CRPs and generated CRPs

To account for the spatial behaviour, in Figure 76 the relationship between Bit-aliasing and Reliability is analysed.

While these analysis show promising results, the current approach has some limitations, specially for cells that are "truly-random", that is, a Bit-aliasing of 0.5. To account for the truly random cells, we propose to extend the HMC model as shown in Figure 77 by adding

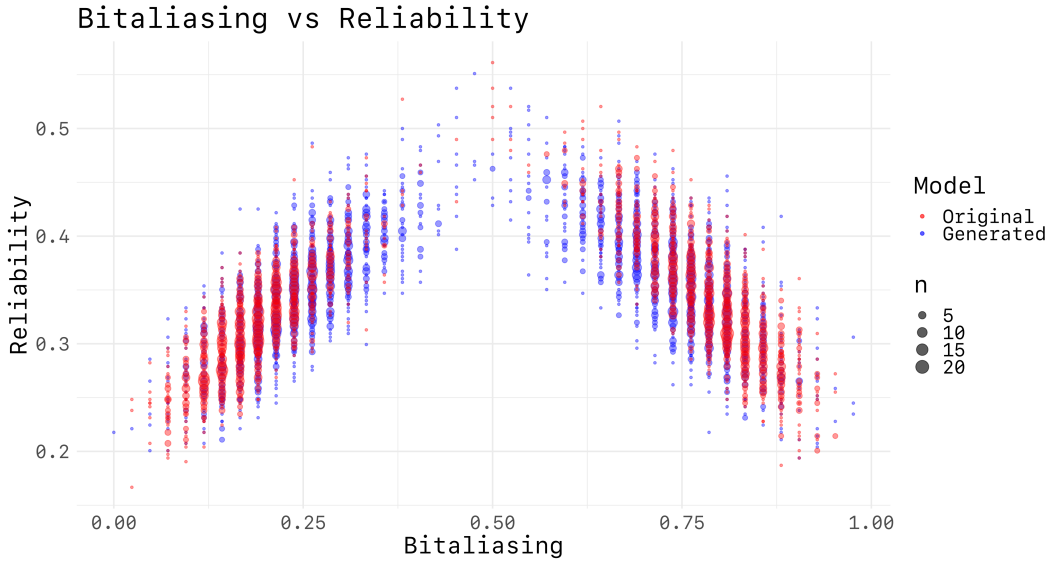


Figure 76: Bit-aliasing vs Reliability of original and generated SRAM CRPs

a new state. Although this is just a proposal, many other modifications can be performed to account for these new cells. Further work will be needed to evaluate a plethora of chains to see which one works better across different types of behaviours and designs.

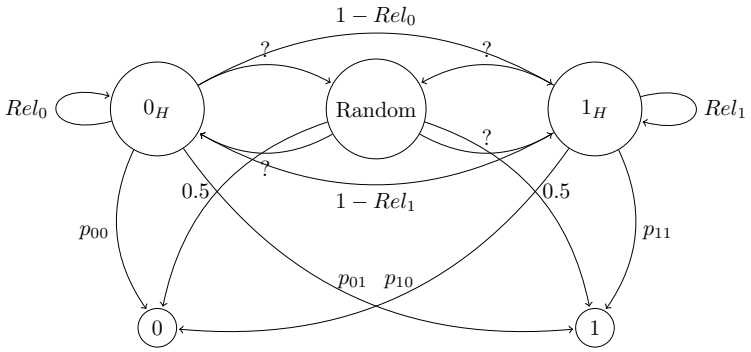


Figure 77: Hidden Markov Chain proposal for response modeling

6.3 Pre-processing and post-processing stages

While the models proposed here may serve as a foundation for a digital twin of a PUF, they do not fully capture the effects of varying external (e.g., noise, voltage) or internal phenomena (e.g., correlation). To address these phenomena, additional pre- and post-processing steps are proposed:

1. Initial assessment of the PUF and data gathering

2. Generation of the twin based on the gathered data
3. Pre-process the transition matrix to adjust for external phenomena
4. Generate a sample of responses
5. Update the state of all cells
6. Post-process the responses to adjust for internal phenomena
7. Repeat from step 3

The post-processing stage can utilize techniques presented in this thesis, primarily the auto-correlation matrix and the joint probabilities derived from it, to account for correlation effects.

The utility of these stages can be seen in the following practical example. Consider that due to certain limitations, the study of a new PUF design can only be performed at a limited range of temperature or voltage values. However, through an in-depth analysis of the randomness source, it is possible to extrapolate changes in reliability under new conditions. These pre- and post-processing stages will provide the tunability in the model that allows researchers to test the PUF under untested environments or to try novel techniques for filtering or reliability improvement.

6.4 Conclusions

The methods described here provide a promising starting point for creating tunable digital twins of certain PUFs. Stochastic cellular automata[2][3] could also be an interesting approach that remains simple enough for practical application. The same pre- and post-processing techniques can be adapted to this new algorithm. However, due to time constraints, these ideas were not explored in this thesis.

For a robust digital twin, advanced statistical techniques such as the aforementioned surrogate models, which can integrate many internal and external parameters, might be more suitable. A significant challenge in creating digital twins is ensuring that they accurately replicate the behaviour of the real PUF. As previously discussed, canonical metrics fail to encapsulate all critical phenomena of a PUF. In other words, we can only observe what we measure. Therefore, further evaluation of these digital twins is necessary to ensure their proper use and accuracy.

Initially, a comparison between different models was planned, but due to time constraints, this was not feasible. The hypothesis underlying this comparison is that each model encapsulates the intricacies of its corresponding PUF design, enabling the differentiation of multiple models from one another. Additionally, since the Hidden MC model generates bits that follow the same nature as the rest of the dataset, bits generated by one model can

[2] A. Agapie *et al.*, 'Probabilistic cellular automata,' *Journal of Computational Biology*, vol. 21, no. 9, pp. 699–708, 2014.
[3] T. Toffoli and N. Margolus, *Cellular automata machines: a new environment for modeling*. MIT press, 1987.

be compared with another dataset to identify discrepancies, indicating that they are “out of place.”

Conclusions

VIII.7

The final chapter of this thesis represents the realization of extensive research and development efforts aimed at understanding and enhancing PUF performance. The core objective was to construct robust mathematical and heuristic models that accurately describe the behaviour of specific PUF designs, providing a comprehensive framework to predict, evaluate, and optimize PUF performance.

An initial analysis of state-of-the-art modeling techniques was presented. Current literature predominantly focuses on machine learning approaches to create predictive models for PUF responses. These models leverage large datasets to train algorithms capable of anticipating PUF behaviour under various conditions. However, the field of mathematical PUF modeling remains largely unexplored, presenting an opportunity to develop foundational models that offer deeper theoretical insights. The work presented in this chapter attempts to fill that gap by establishing the necessary context and methodologies for building such mathematical models.

Following this, the numerical relationships between various canonical PUF metrics are examined. The primary goal is to determine whether Bit-aliasing can serve as a sufficient statistic to describe PUF behaviour, as suggested by some studies. The analysis showed that Bit-aliasing and Uniformity alone can sufficiently predict PUF behaviour, as other metrics can be derived from these, either with statistical or numerical methods.

To validate these numerical relationships, a verification model was derived. This model employs detailed simulations and empirical data to test the robustness of theoretical relationships derived. By studying an ideal system in detail with a Grid Search algorithm, a range of values that the metrics should exhibit given a subset of the entire system was elucidated. This verification process is critical for ensuring the reliability and accuracy of our proposed models, providing a benchmark for further analysis and practical application. Expanding on this work, a series of statistical techniques and heuristics were employed to extrapolate canonical metrics from a small testing dataset. This approach significantly reduces the cost and time required to evaluate PUF performance, as fewer devices and less time are needed for comprehensive testing. By developing efficient statistical models, the behaviour of a PUF can be predicted with high accuracy from limited data, leading to substantial cost savings and efficiency improvements in the development and assessment of PUF designs.

These heuristic analysis set the stage for the creation of a digital twin of a PUF. Various techniques, primarily Markov Chains, were studied to develop a mathematical model that replicates PUF behaviour. Markov Chains, which model systems that transition between states with certain probabilities, are particularly well-suited for capturing the stochastic nature of PUF responses. This digital twin can be easily implemented in software, providing a cost-effective, efficient and tunable means to design and evaluate PUFs. A software-based

approach allows for rapid prototyping and testing, significantly reducing the resources and time needed for PUF evaluation and facilitating more rapid development cycles.

Finally, given the extensive data gathered during this thesis, different digital twins were created and compared to discern the unique characteristics of different PUF families. This comparison involves analysing the response patterns, assessment metrics, and entropy levels of each PUF type to highlight their respective strengths and weaknesses. By understanding these distinctions, different PUF designs can be tailored to specific applications, enhancing their effectiveness and security.

In conclusion, the techniques and models proposed in this chapter pave the way for new evaluation and modeling methodologies that can significantly reduce the cost and time associated with PUF design and testing. These advancements increase the likelihood of broader adoption of PUF technologies across various industries, from secure authentication to anti-counterfeiting measures. However, it is crucial to refine these models with more extensive data and enhanced statistical techniques to ensure they are robust enough for industry acceptance. The proposed frameworks offer a promising foundation for future research and development in PUF technology, aiming to make it more accessible and practical for widespread use.

IX

Conclusions and Perspectives

Security measurements are critical in modern circuits to protect sensitive data from malicious actors. Conventional cryptographic methods rely on private keys for encryption and decryption, but these approaches are vulnerable to physical attacks where sensitive information can be extracted. Physical Unclonable Functions offer a viable alternative by generating unique secrets on the fly, leveraging inherent process variability and eliminating the need for data storage. Among the various PUF designs, Ring Oscillator and SRAM-based PUFs are extensively studied due to their simplicity and ubiquity in modern SoCs.

During the parametric simulations for assessing specific PUF designs, several limitations in available commercial EDA software were identified. To address these challenges, a series of open-source tools were developed, addressing the lack of open tools for AMS design, especially in security contexts. Monaco, the first tool, automates the injection of parameters into SPICE files, enabling extensive simulations despite EDA constraints. However, Monaco's manual design process motivated the creation of NIMPHEL, a Python framework for generating parametrizable circuits, which simplifies and accelerates the design process using programming constructs like loops and arrays.

To validate the simulation results, access to real data is essential for robust evaluation. To address the scarcity of accessible PUF datasets, an open-source platform was created from scratch to extensive SRAM data and sensor readings from microcontrollers. The platform currently gathers data from 84 STM32 microcontrollers, with weekly updates stored in an open-access database to broaden research opportunities by the research community. Additionally, a comprehensive dataset from Infineon, including data from 399 devices with 6 blocks of 256 ROs each, provided invaluable insights for validating simulation hypotheses and exploring new PUF designs.

The performance metrics considered standard for PUF evaluation were reviewed, revealing significant limitations. Several mitigations and new alternative metrics were introduced to address these shortcomings. Real-world data from the SRAMPlatform showed extreme bias and correlation effects that canonical metrics failed to highlight, unlike the new proposals, which accurately describe these effects, underscoring the need for more robust testing methodologies. Given that PUFs are studied across hardware design, cryptography, and material science, these new metrics try to offer valuable information to researchers of these various disciplines, in pursue of better methodologies for PUF development and assessment.

A major focus was placed on the relationship between entropy and reliability in PUFs. In the context of RO-PUFs, the relationship between frequency difference and reliability was exhaustively studied using simulation methodologies. A simulation-based methodology was developed to set reliability thresholds based on frequency differences. Subsequently, a holistic mathematical model accounting for process variability was created, proving useful for optimizing RO-PUF designs. A new design methodology, "Split PUF," was proposed to leverage process variability across different PUF instances, maximizing Entropy yield and Reliability of the underlaying PUF design.

Mathematical modeling of PUFs, a less-explored area, was also tackled. A series of statistical and numerical models were proposed to improve the understanding of RO- and SRAM-based PUF designs, opening new research avenues as they could be generalized to other designs. By leveraging the relationship between PUF metrics, statistical methods for metric extrapolation were proposed and evaluated on existing datasets. This approach reduces the time and cost needed to evaluate PUFs. Additionally, "software" clones of PUFs were proposed using statistical models, facilitating algorithm testing and evaluation. These models provide a robust and cost-effective methodology for assessing PUF performance and aid in their security assessment by justifying PUF behaviour.

The advancements made in this thesis enhance PUF assessment methodologies, addressing limitations in current tools and metrics, and providing new frameworks and models for future research. These contributions are aimed at accelerating the worldwide adoption of PUF technologies, ensuring more secure and reliable implementations across a variety of applications.

Future Work and Perspectives

A short-term objective involves developing a unified testing framework for all PUF families, as discussed in Chapter IX.2. This framework will integrate existing metrics and introduce new ones to create a comprehensive testing suite. The aim is to establish a standardized evaluation methodology, akin to the NIST and Dieharder test suites for random number generators, ensuring consistent and comparable results across different PUF studies.

One critical area for future research is the validation of the statistical models developed for SRAM- and RO-PUFs across a wider spectrum of PUF families. This task requires the collection of extensive data from diverse PUF designs to demonstrate the general applicability of the proposed models, thereby confirming their robustness and reliability across various PUF technologies. Additionally, refinement of the digital twins models proposed is also among the short-term objectives. The addition of more system parameters and novel statistical algorithms could facilitate faster and more reliable validation processes with refined models that better mimic PUF behaviours and peculiarities.

These future research directions aim to enhance PUF assessment methodologies and accelerate their global adoption, ensuring more secure and reliable PUF implementations across a variety of applications.

Mathematical Background

IX.A

A.1 Notation used

In the context of bitstrings and PUF responses, the operator $\oplus$ denotes the XOR operation while the $\odot$ operator denotes the XNOR operation.

The operators $n(S), |S|, \#S$ denote the number of elements in the set S unless otherwise stated. They will be used interchangeably to ease some notations and equations.

In the context of statistical studies, the notation $\overset{iid}{\sim}$ is used to indicate that the random variables $X_1, \ldots, X_n \overset{iid}{\sim} F(\theta)$ are *independent and identically distributed* according to the distribution F with parameters θ. Moreover, in the situations where certain statistical approximations are performed, it will be indicated as $X \dot{\sim} F$, meaning that "X is approximately distributed following F." If the variable X is not distributed according to F, but can be approximated with the distribution function G, the notation $X \dot{\sim} G$ will be used.

A.2 Statistical distribution of physical parameters

Many physical behaviours can be effectively model using a Gaussian distribution due to its natural occurrence in numerous processes governed by random variables and the central limit theorem. For example, the distribution of RO frequencies can be described by a Gaussian distribution.

The behaviour of the Normal distribution is given by its PDF shown in Equation A.1, characterized by the mean μ and by the standard deviation σ.

$$Normal \equiv f(x \mid \mu, \sigma) \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right] \tag{A.1}$$

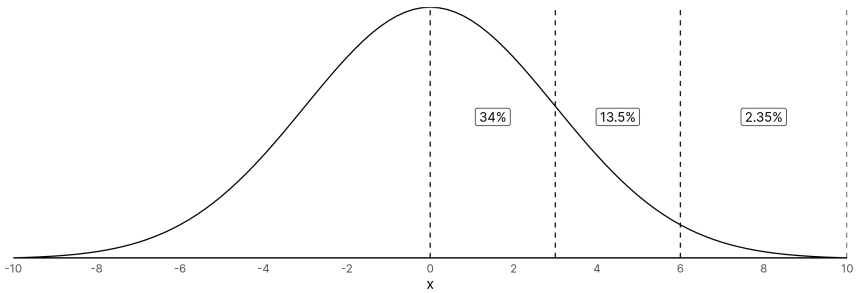


Figure 78: Normal distribution and standard deviation

The standard deviation σ defines the spread of the distribution. About 68% of values drawn from a normal distribution are within one standard deviation away from the mean; about 95% of the values lie within two standard deviations; and about 99.7% are within three standard deviations.

The Cumulative Distribution Function (CDF) of a random Variable X evaluated at x is the probability that X will take a value less than or equal to x. In the case of the normal distribution, it is usually denoted as Φ and it's given by the integral shown in Equation A.2.

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2}\, dt \tag{A.2}$$

For a generic normal distribution with density f, mean μ and variance σ^2, the CDF is

$$\Phi\left(\frac{x-\mu}{\sigma}\right) = \frac{1}{2}\left[1 + erf\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right]$$

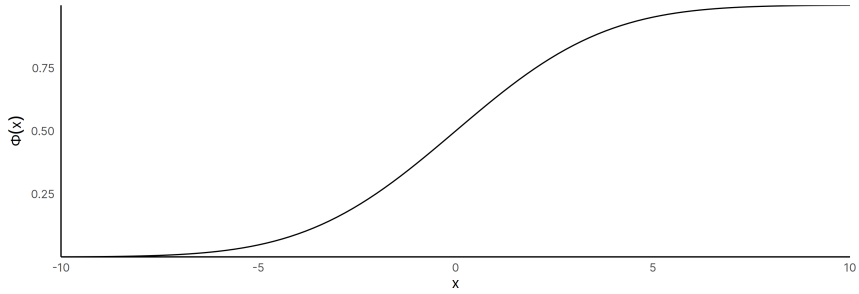


Figure 79: Cumulative Distribution Function of the Normal distribution

The concepts of mean and variance are generalized to all distributions through the expected value and the variance. The expected value of a discrete random variable is defined as

$$\mathrm{E}\left[X\right] = \sum_{i=1}^{\infty} x_i\, p_i,$$

Given a random set of data, the expected value and the variance of the distribution are usually computed with techniques such as Maximum Likelihood Estimation (MLE). For some distributions (i.e the beta distribution), the MLE estimators do not have a close form and they are usually computed numerically. However, in the case of the normal distribution, the MLE estimators for μ and σ^2 are the following.

$$\hat{\mu} = \bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad \hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

An important observation to be made is that in the R programming language (used extensively in this document), and most statistical software, the function `mean()` correctly computes the expected value as shown above. However, the variance is usually computed as the following.

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

Although this difference can have a significance in certain statistical studies, due to the dimensionality of the datasets studied in this manuscript, the difference will be negligible.

There are also a series of properties of the expected value and variance depicted in Equation A.6 that will be proven useful for certain analytical analysis.

$$\mathrm{E}[aX + b] = a\,\mathrm{E}[X] + b \tag{A.3}$$
$$\mathrm{Var}[aX + b] = a^2\,\mathrm{Var}[X] \tag{A.4}$$
$$\mathrm{Var}[X + Y] = Var(X) + Var(Y) + 2\,\mathrm{Cov}[X, Y] \tag{A.5}$$
$$\mathrm{Cov}[aB, bY] = ab\,\mathrm{Cov}[X, Y] \tag{A.6}$$

The covariance $\mathrm{Cov}[X, Y]$, which measures the joint variability of the two random variables X, Y, is computed as Equation A.7, and will also be used extensively in analysis that involve multiple random variables.

$$\mathrm{Cov}[X, Y] = \mathrm{E}[(X - \mathrm{E}[Y])(Y - \mathrm{E}[X])] \tag{A.7}$$

In situations where computing $\mathrm{E}\left[X^2\right]$ is unfeasible or computationally expensive, is it possible to derive it if $\mathrm{Var}[X]$ and $\mathrm{E}[X]^2$ are known as shown in Equation A.8.

$$\mathrm{Var}[X] = \mathrm{Var}\left[X^2\right] - \mathrm{E}[X]^2 \Rightarrow \mathrm{E}\left[X^2\right] = \mathrm{Var}[X] + \mathrm{E}[X]^2 \tag{A.8}$$

A.3 Beta Distribution

Another distribution that is common in multiple scientific disciplines is the Beta distribution. In Bayesian inference, the Beta distribution is the conjugate prior probability distribution for the Bernoulli, binomial, negative binomial, and geometric distributions.

The PDF of the Beta distribution is shown in Equation A.9.

$$f(x \mid \alpha, \beta) = \frac{1}{B(\alpha, \beta)} x^{\alpha-1}(1-x)^{\beta-1} \tag{A.9}$$

The coefficients of the Binomial and Beta distributions are related in the following way.

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{\Gamma(n+1)}{\Gamma(k+1)\Gamma(n-k+1)} \qquad B(a,b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}$$

The expected value and variance of the beta distribution depend purely on α and β.

$$\mathrm{E}[X] = \frac{\alpha}{\alpha+\beta} \qquad \mathrm{Var}[X] = \frac{\alpha \cdot \beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$$

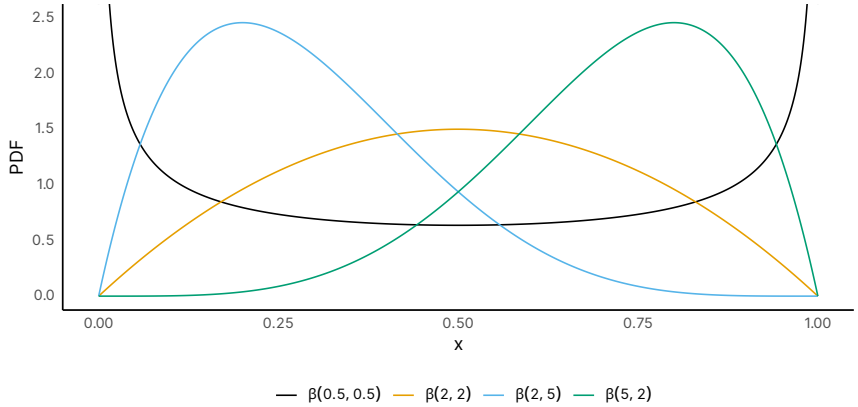


Figure 80: Probability Density Function of the Beta Distribution

The Beta distribution is bounded in $[0, 1]$ while the Normal distribution is bounded in $[-\infty, +\infty]$, so the Beta distribution is more suited to work with probabilities and can better model certain properties of PUFs, as it directly represents the probabilities of bits. Moreover, the relationship between the parameters tells us at first glance the following:

- If $\alpha < \beta$ the distribution is left skewed
- If $\alpha = \beta$ the distribution is symmetric
- If $\alpha > \beta$ the distribution is right skewed

In the case where $\alpha = \beta$, the higher the values, the smaller the variance in for the equivalent normal distribution.

Even though the parameters α and β are usually obtained numerically through MLE, we can estimate them using the method of moments, Given $\overline{x}$ = sample mean, s = sample variance

$$\hat{\alpha} = \overline{x}\left(\frac{\overline{x}(1-\overline{x})}{s} - 1\right) \qquad \hat{\beta} = (1-\overline{x})\left(\frac{\overline{x}(1-\overline{x})}{s} - 1\right)$$

A.4 Gamma Distribution

The Gamma distribution is another versatile continuous probability distribution. In our case, it will prove useful to model waiting times. There are 2 equivalent parametrizations:

- With a shape parameter k and a scale parameter θ
- With a shape parameter $\alpha = k$ and an inverse scale parameter $\beta = 1/\theta$, called a rate parameter.

The PDF of the gamma distribution is shown in Equation A.10.

$$f(x \mid \alpha, \beta) = \frac{x^{\alpha-1} e^{-\beta x} \beta^{\alpha}}{\Gamma(\alpha)} \text{ for } x > 0,\ \alpha, \beta > 0 \tag{A.10}$$

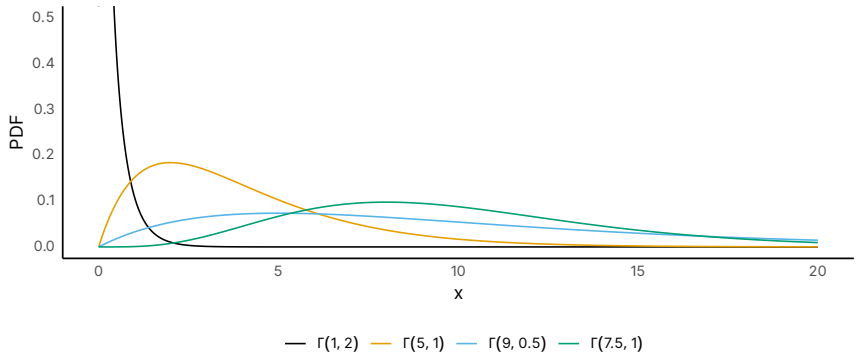


Figure 81: Probability Density Function of the Gamma Distribution

A.5 Laplace Distribution

An uncommon distribution that will be useful in this document, specially in Chapter 4, is the Laplace distribution, parametrized by μ and b as shown in Equation A.11.

$$Laplace \equiv f(x \mid \mu, b) = \frac{1}{2b} \exp\left(-\frac{|x - \mu|}{b}\right) \tag{A.11}$$

The Laplace distribution is tightly related to the Exponential distribution, as we can think of the Laplace distribution as an unfolder version of the Exponential one. In our case, the Laplace distribution will be able to predict the behaviour of the metrics for positive and negative values of PUF metrics for RO-based PUFs.

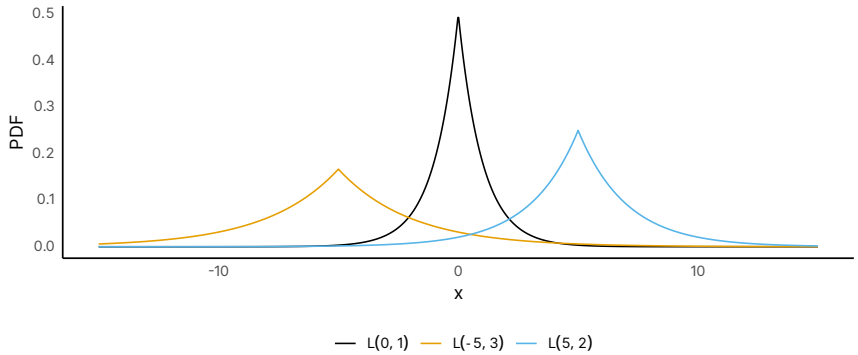


Figure 82: Probability Density Function of the Laplace Distribution

A.6 Binomial Distribution

The Binomial distribution is a fundamental discrete probability distribution that models the number of successes in a fixed number of independent Bernoulli trials, each with the same probability of success. It is characterized by two parameters: n, the number of trials, and p, the probability of success in each trial, as shown in Equation A.12. This distribution is particularly useful in modeling scenarios where there are two possible outcomes, such as flipping a coin, or in digital communications, where it can model the number of bits set to 1 in a bitstring.

$$f(k, n, p) = Pr(k; n, p) = Pr(X = k) = \binom{n}{k} p^k (1-p)^{n-k} \tag{A.12}$$

The binomial operation is defined as follows.

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

And the expected value and variance of the distribution can be computed as follows.

$$\mathrm{E}[X] = np \qquad \mathrm{Var}[X] = np(1-p)$$

In the context of bitstrings, the Binomial distribution helps in understanding and predicting the behavior of binary sequences, which is crucial in fields like coding theory, cryptography, and error detection, where the analysis of binary data and the probability of certain bit patterns occurring play a critical role in designing efficient and reliable systems.

A.7 Poisson Binomial Distribution

The Poisson Binomial distribution generalizes the Binomial distribution to situations where the success probabilities vary across trials. Instead of a fixed probability p for each trial, the Poisson Binomial distribution considers n independent Bernoulli trials with potentially different success probabilities $p_1, p_2, \ldots, p_n$, as shown by its PDF in Equation A.13.

$$Pr(K = k) = \sum_{A \in F_k} \prod_{i \in A} p_i \prod_{j \in A^c} (1 - p_j) \tag{A.13}$$

This distribution is used to model the number of successes when the trials are not identically distributed, making it more flexible than the standard Binomial distribution. However, this flexibility comes with analytical complexity. Calculating probabilities and other characteristics of the Poisson Binomial distribution is much more challenging because it lacks the simple closed-form expressions that the Binomial distribution enjoys. Consequently, working with the Poisson Binomial distribution often requires sophisticated numerical methods or approximations, making it less straightforward to handle in practice.

A.8 Confidence Intervals

Confidence intervals are a crucial concept in statistical inference, providing a range of values within which a population parameter is likely to lie, based on sample data. Instead of offering a single estimate, a confidence interval accounts for variability and uncertainty, offering an upper and lower bound along with a specified confidence level (commonly 95%). This interval gives a sense of the reliability of an estimate and helps in making informed decisions. Confidence intervals are important because they convey not just an estimate, but the precision and uncertainty surrounding that estimate, enabling researchers and practitioners to assess the robustness of their conclusions and to make probabilistic statements about the population parameters being studied.

The most common CI is the Wald CI, where α denotes the confidence level.

$$CI = \bar{x} \pm z_\alpha \frac{s}{\sqrt{n}}$$

For low number of samples the t-statistics is preferred.

$$CI = \bar{x} \pm t_{\alpha/2} \frac{s}{\sqrt{n}}$$

A.9 Statistical Bootstrap

Bootstrap[1] is a statistical technique that uses random sampling with replacement to estimate almost any parameter of statistics from the sampling distribution. Bootstrap will be used in this manuscript to extrapolate certain statistical metrics of PUFs by using a small testing dataset. Due to the robust mathematical foundation of this methodology, it is very simple to way to derive estimates of standard errors and confidence intervals.

For a parameter θ estimated through bootstrap, its standard CI is defined as shown in Equation A.14

$$CI = (\theta^*_{\frac{\alpha}{2}},\ \theta^*_{1-\frac{\alpha}{2}}) \tag{A.14}$$

However, the Pivot Confidence Interval, shown in Equation A.15, tends to be more accurate, so it's usually preferred.

$$CI = \left(2\theta - \theta^*_{1-\frac{\alpha}{2}},\ 2\theta - \theta^*_{\frac{\alpha}{2}}\right) \tag{A.15}$$

A.10 Heteroscedasticity in statistical modeling

When performing Ordinary Least Squares (OLS), Heteroscedasticity can suppose a problem. OLS assumes that the residuals come from a population that has homoscedasticity, which means constant variance. When heteroscedasticity is present in a regression analysis, the results of the analysis become hard to trust. Specifically, heteroscedasticity increases the variance of the regression coefficient estimates, but the regression model doesn't pick up on this. This makes it much more likely for a regression model to declare that a term in the model is statistically significant, when in fact it is not.

Considering the following linear regression equation where the dependent random variable y_i equals the deterministic variable x_i times coefficient β_i plus a random disturbance term ε_i that has mean zero.

$$y_i = x_i\beta_i + \varepsilon_i \quad i = 1, ..., N$$

The disturbances are homoscedastic if the variance of ε_i is a constant σ^2; otherwise, they are heteroscedastic, as shown in Figure 83. In particular, the disturbances are heteroscedastic if the variance of ε_i depends on i or on the value of x_i.

[1] B. Efron, 'Bayesian inference and the parametric bootstrap,' *The annals of applied statistics*, vol. 6, no. 4, p. 1971, 2012.

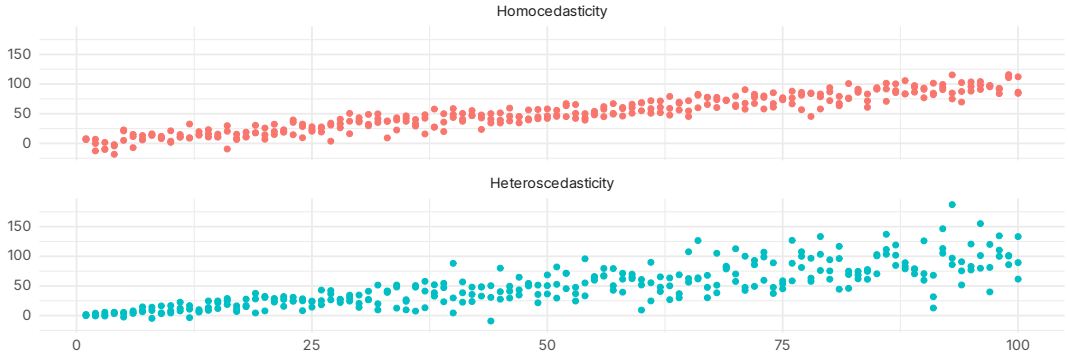


Figure 83: Representation of Homoscedasticity and Heteroscedasticity

Heteroscedasticity will appear when trying to model the behaviour of the PUF for small number of challenges and devices, since the larger the number of devices and challenges the more the results are averaged out and less variability is expected.

A.11 Information Theory

Information theory plays a big role in the analysis of PUFs by providing the mathematical framework to evaluate and quantify the security and reliability of PUFs. Key concepts such as Entropy and Mutual Information are used to measure the randomness and uniqueness of PUF outputs, ensuring that they are sufficiently unpredictable and resistant to cloning or prediction attacks. Additionally, information theory aids in analyzing error rates and the robustness of PUF responses in different environmental conditions, which is critical for their practical deployment in secure authentication and cryptographic applications. By applying information-theoretic principles, researchers can rigorously evaluate the performance and security properties of PUFs, ensuring their effectiveness in real-world security systems.

The Hamming Weight of a vector is defined as the number of symbols different from the zero-symbol. In the case of bitstrings or PUF responses, it represents the number of 1s. This function as also known as `popcount`, or population count, specially in the field of computer science.

$$\text{Hamming Weight}(X) = HW(X) = \texttt{popcount}(X) = \sum_{x \in X} x \tag{A.16}$$

The Hamming Distance of two vectors of equal length is the number of values at equal positions that differ.

$$\text{Hamming Distance}(X, Y) = HD(X, Y) = \sum_{i}^{n(X)} X_i \oplus Y_i \qquad \{n(X) = n(Y)\} \tag{A.17}$$

More commonly we use the *fractional* or *normalized* Hamming Weight and Hamming Distance, which normalize the results with respect to the input size.

$$HW_F(X) = \frac{HW(X)}{n(X)} \qquad HD_F(X,Y) = \frac{HD(X,Y)}{n(X)}$$

A.11.1 Entropy

Even though there are multiple definitions of Entropy, Shannon Entropy[2] is the most common one in the context of cryptography. Given a discrete random variable X, which takes values in the alphabet $\mathcal{X}$ and is distributed according to $p : \mathcal{X} \to [0,1]$, its Shannon Entropy is computed as follows:

$$\mathrm{H}(X) = -\sum_{x \in \mathcal{X}} p(x) \log p(x) \tag{A.18}$$

Although other bases can be used, we use log base 2, aka Shannons, as it,s the norm in information theory or when working with bits. Also, by convention $1 \log 1 = 0$ and $0 \log 0 = 0$. The latter is because $\lim_{x \to 0^+} x \log x = 0$

For binary vectors, it can be summarised in the following way, where $p = P(1)$ and $q = 1 - p$

$$\mathrm{H}(X) = -\left(p \log_2 p + q \log_2 q\right)$$

A.11.2 Joint Entropy

The concept of Entropy of a random variable can be extended to multiple variables and takes the name of Joint Entropy.

$$\mathrm{H}(X_1, \ldots, X_n) = -\sum_{x_1 \in \mathcal{X}_1} \cdots \sum_{x_n \in \mathcal{X}_n} P(x_1, \ldots, x_n) \log_2 [P(x_1, \ldots, x_n)]$$

Although numerous random variables can be computed at the same time, here it will be used to analyse pairs of bits to study the relationship between the two.

$$\mathrm{H}(X,Y) = -\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} P(x,y) \log_2 [P(x,y)]$$

[2] F. M. Reza, *An introduction to information theory*. Courier Corporation, 1994.

A.11.3 Kullback-Leibler divergence

The Kullback-Leibler divergence measures how different two distributions are. It the field of ML is commonly known as Information Gain since it refers to the amount of information gained about a random variable from observing another random variable. Given a reference distribution $Q(x)$ and our distribution $P(x)$, the Kullback-Leibler divergence is computed as depicted in Equation A.19.

$$D_{KL}(P(x) \parallel Q(x)) = \sum_{x \in X} P(x) \log\left(\frac{P(x)}{Q(x)}\right) = -\sum_{x \in X} P(x) \log\left(\frac{Q(x)}{P(x)}\right) \tag{A.19}$$

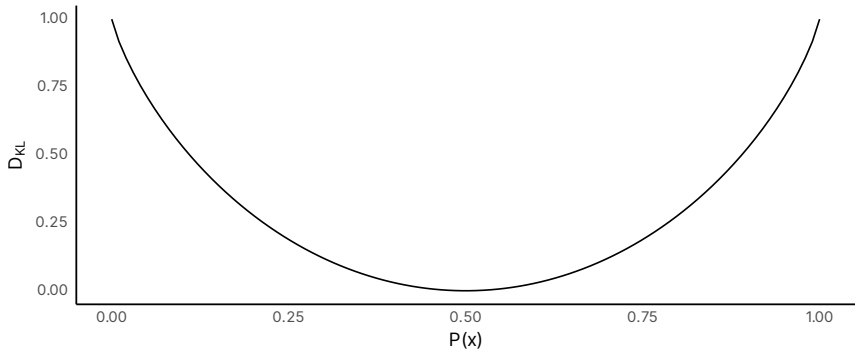


Figure 84: Kullback-Leibler divergence between P(x) and Q(x)

It's important to state that this function is not symmetric, that is, if we swap $P(x)$ and $Q(x)$ we obtain different results

A.11.4 Mutual Information

Mutual information is a fundamental concept in information theory that measures the amount of information two random variables share; it quantifies the reduction in uncertainty about one variable given knowledge of the other. In the context of PUFs, low mutual information between responses indicates high unpredictability and distinctiveness, crucial for ensuring that PUFs are resistant to cloning and prediction attacks. This metric helps in validating the security properties of PUFs, ensuring that each response is unique and independent, thus reinforcing the robustness of PUF-based authentication systems.

$$I(X;Y) = D_{KL}(P_{(X,Y)} \parallel P_X \otimes P_Y)$$

Where D_{KL} is the Kullback-Leibler divergence and $P_X \otimes P_Y$ is the outer product distribution which assigns probability $P_Y(x) \cdot P_Y(y)$ to each (x, y)

$$I(X;Y) = \sum_{y \in \mathcal{Y}} \sum_{x \in \mathcal{X}} P_{(X,Y)}(x,y) \log \left(\frac{P_{(X,Y)}(x,y)}{P_X(x) P_Y(y)} \right)$$

$$I(X;Y) \equiv \mathrm{H}(X) - \mathrm{H}(X \mid Y) \equiv \mathrm{H}(Y) - \mathrm{H}(Y \mid X) \tag{A.20}$$
$$\equiv \mathrm{H}(X) + \mathrm{H}(Y) - \mathrm{H}(X,Y) \equiv \mathrm{H}(X) + \mathrm{H}(Y) - \mathrm{H}(X,Y) \tag{A.21}$$
$$\equiv \mathrm{H}(X,Y) - \mathrm{H}(X \mid Y) - \mathrm{H}(Y \mid X) \tag{A.22}$$

A.12 Markov Chains

A Markov Chain (MC) is a mathematical system that undergoes transitions from one state to another within a finite or countable state space, where the probability of each subsequent state depends only on the current state and not on the sequence of events that preceded it. This property, known as the Markov property, simplifies the analysis of stochastic processes. Markov Chains are invaluable in statistics because they provide a robust framework for modeling a wide range of time-dependent random phenomena, such as stock prices, weather patterns, and population genetics. Their simplicity and tractability make them useful for simulating and understanding complex systems.

For example, a non-biased coin with probabilities of head and tails of 0.5, 0.5 respectively, is represented as shown in Figure 85.

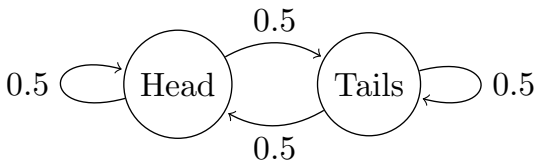


Figure 85: Simple Markov Chain

The probabilities of transitions between the different states are commonly represented using a Transition Probability Matrix P. Each entry in the matrix P represents the probability of going from the current state to the desired state.

$$(P)_{i,j} = P(X_{n+1} = j \mid X_n = i)$$

Which is the case of the coin, the transition matrix is represented as

$$P = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix}$$

In order for the Transition matrix to represent valid probability distributions, it is required that either every row or column sums up to up to 1, depending on how the matrix is used. The state of the markov chain is represented by a vector where each entroy represents the probability of the chain of being at each state. To advance one state, the state vector is multiplied by the P matrix to obtain the new state vector.

One important property of Markov Chains is that they are "memory-less". That is, the transition probabilities of future states only depend on the current state and not on previous states.

$$P(X_{n+1} = s_{n+i} \mid X_n = s_n, X_{n-1} = s_{n-1}, X_{n-2} = s_{n-2}, \ldots) = P(X_{n+1} = s_{n+1} \mid X_n = s_n)$$

```
simulator( 'spectre )

design(  "/home/users/vinagres/projet_cmos065_58/simulation/CMOS_RO_65/spectre/schematic/netlist/netlist")
resultsDir( "/home/users/vinagres/projet_cmos065_58/simulation/CMOS_RO_65/spectre/schematic" )
modelFile(
    '("/home/users/vinagres/projet_cmos065_58/corners.scs" "")
)
analysis('tran ?stop "101n" ?step "0.1p" ?maxstep "0.1p"
 ?max_consecutive_minstep 10e-20 )
envOption(
        'analysisOrder  list("tran")
)
temp( 29 )
out = outfile("/home/users/vinagres/PhD/CMOS_RO_65/freq.txt" "w")

run()

Freq_OutStable = frequency(clip(v("/OutStable" ?result "tran") 2e-08 2))
fprintf(out "%.10g\n" value(Freq_OutStable))

exit()
```

Listing B.1: Example Ocean Script. The command exit() is needed when executing ocean scripts from the terminal or ocean will wait for user input

```
simulator( 'spectre )
design( "/home/users/vinagres/projet_cmos065_58/simulation/CMOS_RO_NAND/spectre/schematic/netlist/netlist")
resultsDir( "/home/users/vinagres/projet_cmos065_58/simulation/CMOS_RO_NAND/spectre/schematic" )

modelFile(
    '("/home/users/vinagres/projet_cmos065_58/corners.scs" "")
)
{% if props.noise_runs -%}
analysis('tran ?stop "{{props.time_sim}}n"  ?tranNoise "Transient Noise"
 ?noisefmax "{{props.noise_fmax}}"
         ?noisefmin "{{props.noise_fmin}}" ?noisescale ""  ?noisetmin ""
         ?noiseupdate "" ?tranNoiseMultiRuns "Multiple Runs"  ?noiseruns
          "{{props.noise_runs}}"
         ?noiseonoff ""  ?noiseinst "/I31 /I32 /I33 /I34"  ?step
          "{{props.time_step}}p"
         ?maxstep "{{props.time_step}}p"  ?minstep "{{props.time_step}}p"  )
{% else -%}
analysis('tran ?stop "{{props.time_sim}}n" ?step "{{props.time_step}}p"
         ?maxstep "{{props.time_step}}p"  ?minstep "{{props.time_step}}p" )
{% endif -%}

envOption(
        'analysisOrder  list("tran")
)
```

```
option('temp  "{{props.temp}}")
temp( {{props.temp}} )
run()

out = outfile("{{props.freq_results}}" "w")

{% if props.noise_runs -%}
{%- for run in range(1, props.noise_runs) %}
Freq_Output = frequency(value(getData("/OutStable" ?result "tran") "Iteration"
 ↪  {{run}}))
fprintf(out "%.10g\n" value(Freq_Output))
{% endfor %}
{% else -%}
Freq_Output = frequency(clip(v("/OutStable" ?result "tran") 2e-08 2))
fprintf(out "%.10g\n" value(Freq_Output))
{% endif -%}

exit()
```

Listing B.2: Example Ocean Script with template placeholders

```
`include "constants.vams"
`include "disciplines.vams"

module counter (ro_1);
    electrical ro_1;
    integer log_fp;

    integer count;
    parameter real step = 5e-9;
    parameter real signal_thresh = 0.5;
    parameter string log_path = "/path/to/results.csv";

    integer en_count = 0;

analog begin
        // Initial configuration
        @(initial_step) begin
            count = 0;
            log_fp = $fopen(log_path, "a");
            $fwrite(log_fp, "time,vout,count\n");
        end

        @(final_step) begin
            $fclose(log_fp);
        end

        @(cross(V(ro_1) - signal_thresh, +1)) begin
              if (en_count == 1) begin
                count = count + 1;
                en_count = 0;
              end
        end

        @(cross(V(ro_1) - signal_thresh, -1)) begin
                en_count = 1;
        end

        @(timer(20e-09)) begin
```

```
                count = 0;
        end

        @(timer(20e-09, step)) begin
            $fwrite(log_fp, "%g,%g,%d\n", $abstime, V(ro_1), count);
        end
end

endmodule
```

Listing B.3: Verilog-A implementation of the counter that measures the frequency of a single Ring Oscillator

```
module observer(p, n);
    input p, n;
    // electrical vp, vn, ip, in;

    // types definition
    // Distance comp;
    Resistance p, n;

    parameter real step = 10e-9;
    parameter string path = "/path/to/results.csv";
    parameter string mode = "a";

    // output file descriptor
    integer fp_tt;

    // Analog behaviour
    analog begin
        @(initial_step) begin
            fp_tt=$fopen(path, mode);
        end

        @(timer(0, step)) begin
            $fwrite(fp_tt,"%.10g,%g\n", $abstime, Ohm(p, n));
        end

        @(final_step) begin
            $fclose(fp_tt);
        end
    end
endmodule
```

Listing B.4: Verilog-A implementation of the observer that stores parameters of interest at the desired timesteps

```
jobs:
  - name: Simulation
    dir: "/home/users/vinagres/PhD/CMOS_RO_Parasitic"
    props:
      cadence_project:
       ↪  "/home/users/vinagres/projet_cmos065_58/simulation/CMOS_RO_NAND"
      nstages: 4
      nom_wp: 0.27 # In micrometers
      nom_wn: 0.135
      nom_l: 0.06 # In micrometers
```

```yaml
      nom_vthp: -0.43
      nom_vthn: 0.4223
      counter_results:
        - "/home/users/vinagres/PhD/CMOS_RO_Parasitic/results/counter_1.csv"
        - "/home/users/vinagres/PhD/CMOS_RO_Parasitic/results/counter_2.csv"
      freq_results:
       ↪  "/home/users/vinagres/PhD/CMOS_RO_Parasitic/results/freq.csv"
      time_sim: 521 # In nanoseconds
      time_step: 3 # In nanoseconds
      temp: 27
      vdd: 1.0
      values: [0, 1, 2, 3]
      values2: [4, 5, 6, 7]
    templates:
      - "template.netlist:{{cadence_project}}/spectre/schematic/netlist/netlist"
      - "template.ocn:script.ocn"
    iters:
      from: 245
      to: 20000
    steps:
      - ocean -nograph -restore $HOME/PhD/CMOS_RO_Parasitic/script.ocn
      - $HOME/PhD/CMOS_RO_Parasitic/analize_results.py {{iter}}
       ↪  $HOME/PhD/CMOS_RO_Parasitic
```

Listing B.5: Example configuration in YAML for Monaco

Figure 86: Picture of the deployed platform used to gather the data

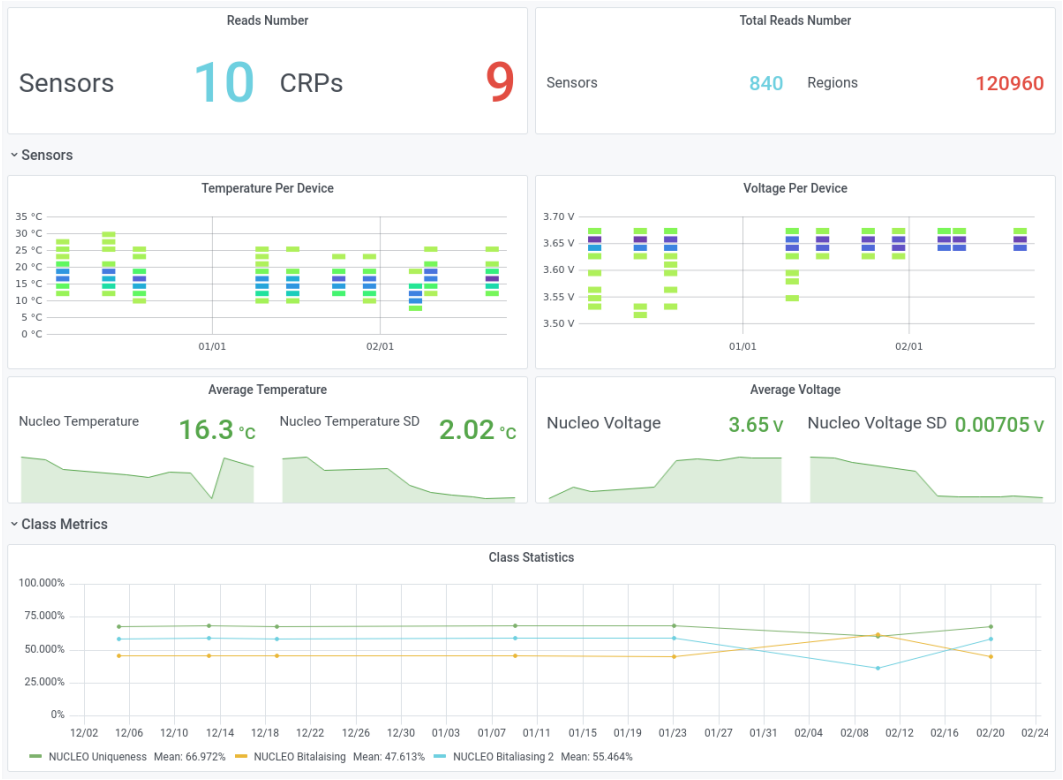


Figure 87: Picture of Grafana

Figure 88: Picture of Grafana

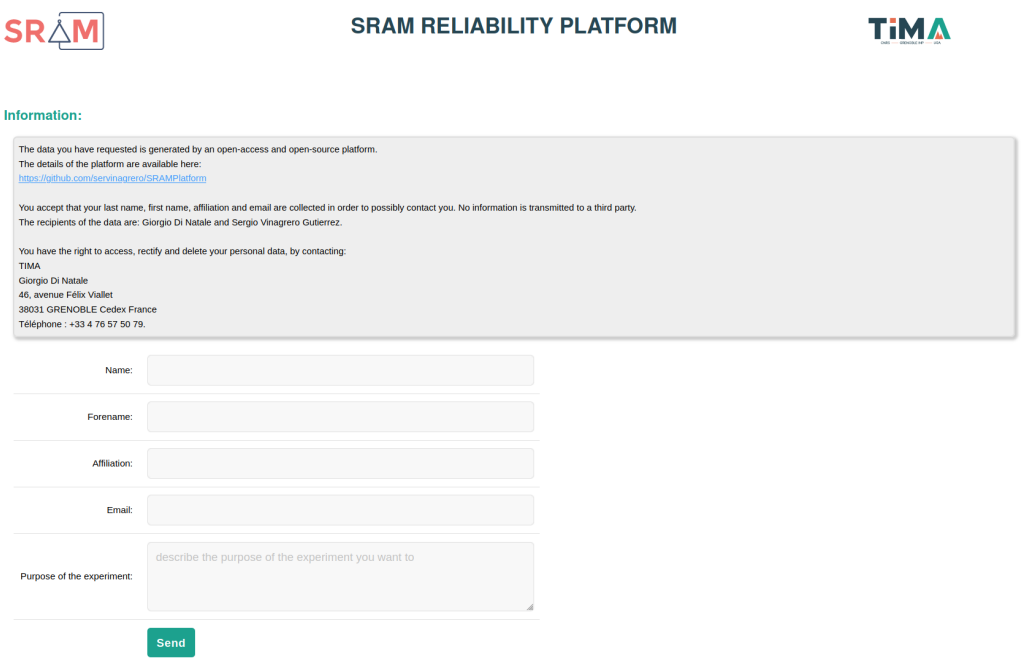


Figure 89: Picture of the PUF4IOT website

```python
class STM32Reader(Reader):
    """Reader implementation for STM32 boards.

    The functionaly of the reader is implemented in the methods called `handle_{command}`.

    Attributes:
        name: Descriptive name of the Reader.
        devices: List of managed devices.
        port: State of the devices and the serial port.
    """

    def __init__(self, board_type: str, port: str, baudrate: int, data_size: int):
        super(STM32Reader, self).__init__(board_type)
        self.devices: List[Device] = []
        self.name = board_type
        self.data_size = data_size

        port_path = Path(port)
        if not port_path.exists():
            print(f"Port {port_path} does not exist")
            sys.exit(1)

        ser = Serial(port_path.as_posix(), baudrate, timeout=None)
        self.port = {"state": "ON", "serial": ser, "path": port_path}


    def handle_sensors(self, props: Dict[str, Any], logger, db_session):
        """Register the devices connected to the reader.
        Args:
            props: Dictionary containing the message from the dispatcher.
        Returns:
            Dictionary with the status of the operation and metadata if needed.
        """
        if self.port["state"] == "OFF":
            raise CommandError("Serial port is off. Turn on the serial port first")
        if not self.devices:
            raise CommandError("No devices managed")
        for dev in self.devices:
            packet = Packet(self.data_size)
            packet.with_command(Command.SENSORS)
            packet.with_uid(dev.uid)
            packet.craft()
            self.send(packet.to_bytes())
            res = next(iter(self.receive()), None)
            if res is None:
                logger.error(f"Problem reading sensors for device {dev}")
                continue
            if not packet.check_crc() or packet.command == Command.ERR:
                logger.warning(f"Packet {packet!s} for device {dev} is corrupted")
                continue
            sensors_data = res.extract_sensors()
            logger.results(
                json.dumps(
                    {
                        "device": {"uid": dev.uid, "pic": dev.pic},
                        "temperature": sensors_data["temperature"],
```

```
                        "voltage": sensors_data["voltage"],
                    }
                )
            )

        db_session.add(
            Sensor(
                uid=format_uid(res.uid),
                board_type=self.name,
                temperature=sensors_data["temperature"],
                voltage=sensors_data["voltage"],
            )
        )
        db_session.commit()
```

Listing C.1

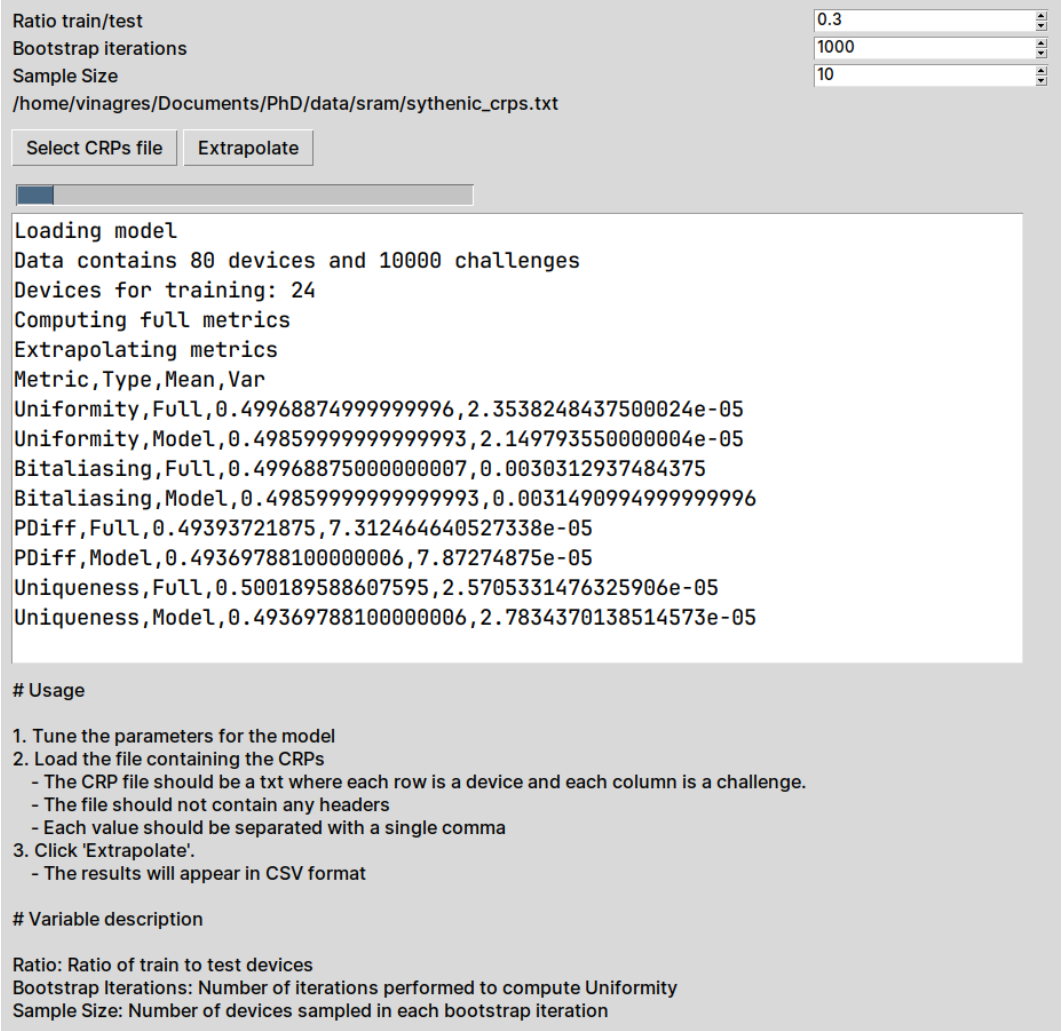


Figure 90: Python interface to extrapolate PUF metrics given a test dataset

```python
#!/usr/bin/env python3

"""
PUF Metric Extrapolation

Prototype app that performs the extrapolation of metrics given a training set of CRPs

References:
https://en.wikipedia.org/wiki/Bootstrapping_(statistics)#Bayesian_bootstrap
https://github.com/gdmarmerola/random-stuff/blob/master/bayesian_bootstrap/The_bayesian_bootstrap.ipynb
"""

from pathlib import Path
from rich import print as pprint

import numpy as np


def uniqueness(crps):
    """Compute uniqueness of the CRPs
    Args:
        crps: Matrix containing the CRPs

    Returns:
        A vector containing all uniqueness values
    """
    devs, challenges = crps.shape
    num_pairs = (devs * (devs - 1)) // 2
    uniq = np.zeros(num_pairs)
    count = 0
    for i in range(devs - 1):
        for j in range(i + 1, devs):
            uniq[count] = np.count_nonzero(crps[i,] != crps[j,]) / challenges
            count += 1
    return uniq


def compute_metrics(crps):
    """Compute all metrics of a set of CRPs

    Args:
        crps: Matrix containing the CRPs

    Returns:
        Dictionary containig the mean and variance of 'uniformity', 'bitaliasing', 'pdiff' and 'uniqueness'
    """
    unif = crps.mean(1)
    ba = crps.mean(0)
    pdiff = 2 * ba * (1 - ba)
    uniq = uniqueness(crps)
    return {
        "uniformity": [unif.mean(), unif.var()],
        "bitaliasing": [ba.mean(), ba.var()],
        "pdiff": [pdiff.mean(), pdiff.var()],
        "uniqueness": [uniq.mean(), uniq.var()],
    }

```

```python

class PUFModel:
    """PUF Metric Extrapolator

    Attributes:
        crps: Matrix containing the CRPs
    """

    def __init__(self, ratio, iters, sample_size):
        self.crps = None
        self.ratio = ratio
        self.iters = iters
        self.sample_size = sample_size

    def load_file(self, path):
        """Load CRPs from a text file"""
        if not Path(path).exists():
            raise ValueError(f"File {path} does not exist")
        self.crps = np.loadtxt(path)

    def split_crps(self, ratio=None):
        """Split the CRPs into the training and test datasets

        Args:
            ratio: Ratio

        Returns:
            A tuple containing the training and set crps
        """
        if self.crps is None:
            raise ValueError("No CRPs provided")

        ratio = ratio or self.ratio
        if ratio < 0 or ratio > 1:
            raise ValueError("Ratio should be in the range [0, 1]")

        devices, _ = self.crps.shape
        indices = np.random.permutation(devices)
        num_train = int(ratio * devices)
        train_idx, test_idx = indices[:num_train], indices[num_train:]
        train_crps, test_crps = self.crps[train_idx, :], self.crps[test_idx, :]
        return train_crps, test_crps

    def bootstrap(
        self, crps, iters, is_bimodal: bool, is_aliased: bool, sample_size=None
    ):
        """Perform bootstrap to compute metrics

        All metrics are computed but we are specially interested in
        ↪ Var[Uniformity]

        Yes, the number of samples drawn in each bootstrap iteration $n$ can be
        ↪ less than or greater than the size of the original dataset. This
        ↪ flexibility allows for different types of resampling techniques in
        ↪ bootstrap.

```

```
        1. **Less than the size of the dataset  $n < N$**: This is known as sub-sampling or Monte Carlo bootstrap. In this case, you randomly sample a subset of the original dataset in each iteration. The resulting bootstrap samples may not represent the full variability of the original dataset, especially if important patterns or characteristics are lost in the smaller sample size. However, sub-sampling can be computationally efficient and may still provide useful estimates, particularly if the dataset is very large.

        2. **Equal to the size of the dataset $n = N$**: This is the standard approach in bootstrap resampling. Each bootstrap sample is the same size as the original dataset. This method provides a robust estimate of the sampling distribution of a statistic and is widely used in practice.

        3. **Greater than the size of the dataset $n > N$**: This is known as the smoothed bootstrap or over-sampling. In this case, you draw more samples than the size of the original dataset by sampling with replacement. This can introduce additional variability into the bootstrap estimates, potentially leading to wider confidence intervals or increased bias in the estimates.

        The choice of $n$ influences the results of the bootstrap in terms of the accuracy, bias, and computational efficiency of the estimates. Generally, using $n = N$ is preferred when possible as it provides the most accurate estimates, but sub-sampling or over-sampling may be necessary in certain situations depending on the characteristics of the dataset and the goals of the analysis.

        Args:
            crps:
            iters: Number of iteranios to perform
            sample_size: Number of devices to sample in each iteration
            is_extreme: Bitaliasing presents extreme values (0 or 1)

        Returns:
            Bootstrapped metrics
        """
        num_devs, _ = crps.shape
        ba = np.zeros((2, iters))
        pdiff = np.zeros((2, iters))
        ba_sqrd = np.zeros((2, iters))
        unif = np.zeros((2, iters))
        sample_size = sample_size or num_devs

        for i in range(iters):
            idx = np.random.choice(range(num_devs), sample_size, replace=True)
            crps_iter = crps[idx, :]

            unif_vec, ba_vec = crps_iter.mean(1), crps_iter.mean(0)
            pdiff_vec = 2 * ba_vec * (1 - ba_vec)
            ba_sqrd_vec = ba_vec**2

            unif[0, i] = unif_vec.mean()
            unif[1, i] = unif_vec.var()
            ba[0, i] = ba_vec.mean()
            ba[1, i] = ba_vec.var()
            pdiff[0, i] = pdiff_vec.mean()

            if is_aliased:
                pdiff[1, i] = pdiff_vec[pdiff_vec > 0].var()
            else:
```

```
                pdiff[1, i] = pdiff_vec.var()

            ba_sqrd[0, i] = ba_sqrd_vec.mean()
            ba_sqrd[1, i] = ba_sqrd_vec.var()

        return {
            "unif_mean": unif[0, :].mean(),
            "unif_var": unif[1, :].mean(),
            "ba_mean": ba[0, :].mean(),
            "ba_var": ba[1, :].mean(),
            "pdiff_mean": pdiff[0, :].mean(),
            "pdiff_var": pdiff[1, :].mean(),
            "ba_sqrd_mean": ba_sqrd[0, :].mean(),
            "ba_sqrd_var": ba_sqrd[1, :].mean(),
        }

    def __handle_unimodal(self, boot, cov):
        """Handle unimodal case"""
        ba_mean, ba_var = boot["ba_mean"], boot["ba_var"]
        pdiff_mean, pdiff_var = boot["pdiff_mean"], boot["pdiff_var"]
        pdiff_mean = 2 * (ba_mean - ba_var * self.ratio - ba_mean**2)
        pdiff_var = 4 * (ba_var * self.ratio + boot["ba_sqrd_var"] - 2 * cov)
        return {
            "uniformity": [ba_mean, boot["unif_var"]],
            "bitaliasing": [ba_mean, ba_var * self.ratio],
            # "pdiff": [pdiff_mean, pdiff_var * self.ratio**4],
            "pdiff": [pdiff_mean, pdiff_var],
            "uniqueness": [pdiff_mean, pdiff_var],
        }

    def __handle_bimodal(self, boot, cov):
        """Handle bimodal case"""
        ba_mean, ba_var = boot["ba_mean"], boot["ba_var"]
        pdiff_mean, pdiff_var = boot["pdiff_mean"], boot["pdiff_var"]
        return {
            "uniformity": [ba_mean, boot["unif_var"]],
            "bitaliasing": [ba_mean, ba_var],
            "pdiff": [pdiff_mean, pdiff_var],
            "uniqueness": [pdiff_mean, pdiff_var / 4],
        }

    def extrapolate(self):
        """

        We should handle:
            - Bimodal distributions of Bitaliasing
            - Extreme bitaliasing values (ba = 0 or ba = 1)
        """
        assert not self.crps is None
        train_crps, test_crps = self.split_crps()

        ba = self.crps.mean(0)
        # ba_sqrd = ba**2
        ba_train = train_crps.mean(0)
        # ba_sqrd_train = ba_train**2
        ba_test = test_crps.mean(0)
        # ba_sqrd_test = ba_test**2
        # print(f"{ba.var() = }")
        # print(f"{ba_sqrd.var() = }")
        # print(f"{ba_train.var()*self.ratio = }")
        # print(f"{ba_sqrd_train.var()*self.ratio = }")
        # print(f"{ba_test.var()*(1-self.ratio) = }")
```

```
        # print(f"{ba_sqrd_test.var()*(1-self.ratio) = }")
        # # Apparently we can approximate Var[BA^2] to Var[BA]

        # Detect if there is extreme bitaliasing
        is_aliased = (ba ⩾ 0.99).any() or (ba ⩽ 0.01).any()

        # Quick and dirty bimodal detection
        ba_right = ba[ba ⩾ 0.5].mean()
        ba_left = ba[ba ⩽ 0.5].mean()
        is_bimodal = (ba_right - ba_left) ⩾ 0.2

        print(f"{is_aliased = }")
        print(f"{is_bimodal = }")

        boot = self.bootstrap(
            train_crps, self.iters, is_bimodal, is_aliased, self.sample_size
        )
        pprint(boot)

        if is_bimodal:
            # Here we expect different values for cov
            cov = np.cov(ba, ba**2)[0, 1]
            cov_left = np.cov(ba[ba < 0.5], ba[ba < 0.5] ** 2)[0, 1]
            cov_right = np.cov(ba[ba > 0.5], ba[ba > 0.5] ** 2)[0, 1]
            print(f"{cov_left=},{cov_right=}")
            print(f"{cov=}")

            ba_mean, ba_var = boot["ba_mean"], boot["ba_var"]
            # ba_mean, ba_var = ba_train.mean(), ba_train.var()*self.ratio
            pdiff_mean = 2 * (ba_mean - ba_var - ba_mean**2)
            pdiff_var = boot["pdiff_var"]
            # pdiff_var = 4*(ba_var + boot['ba_sqrd_var'] - 2*cov)/4
            return {
                "uniformity": [ba_mean, boot["unif_var"]],
                "bitaliasing": [ba_mean, ba_var],
                "pdiff": [pdiff_mean, pdiff_var],
                "uniqueness": [pdiff_mean, pdiff_var / (2 * np.sqrt(2))],
            }

        else:
            # Here we expect the same values for the 3 Cov
            # cov = np.cov(ba, ba**2)[0, 1]
            # cov_train = np.cov(ba_train, ba_train**2)[0, 1]
            # cov_test = np.cov(ba_test, ba_test**2)[0, 1]

            # Apparently bootstrap makes bitaliasing worse here
            # ba_mean, ba_var = boot["ba_mean"], boot["ba_var"]*self.ratio

            ba_mean, ba_var = ba_train.mean(), ba_train.var() * self.ratio
            pdiff_mean, pdiff_var = boot["pdiff_mean"], boot["pdiff_var"]
            pdiff_mean = 2 * (ba_mean - ba_var - ba_mean**2)
            # pdiff_var = 4*ba_var - (2*cov_train*self.ratio) +
             ↪  (4*ba_sqrd_train.var()*self.ratio)
            pdiff_var = 0.025 * ba_var

            return {
                "uniformity": [ba_mean, boot["unif_var"]],
                "bitaliasing": [ba_mean, ba_var],
                "pdiff": [pdiff_mean, pdiff_var],
                "uniqueness": [pdiff_mean, pdiff_var / (2 * np.sqrt(2))],
            }
```

Listing D.1: Python module to extrapolate PUF metrics given a test dataset

```
#!/usr/bin/env python3

import threading
import time
from string import Template
import tkinter as tk
from tkinter import filedialog
from tkinter import font
from tkinter import ttk
from tkinter import messagebox as mb

from rich import print as pprint

import pufmodel

help = """
# Usage

1. Tune the parameters for the model
2. Load the file containing the CRPs
    - The CRP file should be a txt where each row is a device and each column is a challenge.
    - The file should not contain any headers
    - Each value should be separated with a single comma
3. Click 'Extrapolate'.
    - The results will appear in CSV format

# Variable description

Ratio: Ratio of train to test devices
Bootstrap Iterations: Number of iterations performed to compute Uniformity
Sample Size: Number of devices sampled in each bootstrap iteration
""".strip()


def load_file():
    """Load the txt file with the CRPs"""
    path = filedialog.askopenfile(
        title="Load CRPs", initialdir="/home/vinagres/Documents/PhD/data/sram"
    )
    if path:
        file_var.set(path.name)


def extrapolate(metrics):
    text.insert(tk.END, "Loading model\n")

    model = pufmodel.PUFModel(ratio_var.get(), iters_var.get(), sample_size_var.get())
    model.load_file(file_var.get())

    devices, challenges = model.crps.shape
    text.insert(
        tk.END, f"Data contains {devices} devices and {challenges} challenges\n"
    )
    text.insert(tk.END, f"Devices for training: {int(ratio_var.get() * devices)}\n")

```

```
    text.insert(tk.END, "Computing full metrics\n")
    metrics["Full"] = pufmodel.compute_metrics(model.crps)

    text.insert(tk.END, "Extrapolating metrics\n")
    metrics["Model"] = model.extrapolate()


def start_model():
    """
    First we handle some errors
    """
    if not file_var.get():
        mb.showerror("Problem", "No CRP file selected")
        return
    work_thread = threading.Thread(target=exec_model)
    work_thread.start()


def exec_model():
    """Start the model to extrapolate the metrics"""
    pb.configure(value=0, maximum=100)
    pb.start()
    text.configure(state="normal")
    text.delete(1.0, tk.END)  # Clear previous text

    metrics = {}
    work_thread = threading.Thread(target=extrapolate, args=(metrics,))
    work_thread.start()
    work_thread.join()

    # Results are printed in CSV format
    text.insert(tk.END, "Metric,Type,Mean,Var\n")

    pprint(metrics)
    for metric in ["Uniformity", "Bitaliasing", "PDiff", "Uniqueness"]:
        mean, var = metrics["Full"][metric.lower()]
        text.insert(tk.END, f"{metric},Full,{mean},{var}\n")
        mean, var = metrics["Model"][metric.lower()]
        text.insert(tk.END, f"{metric},Model,{mean},{var}\n")

    text.configure(state="disabled")
    pb.stop()
    pb.configure(value=0, maximum=0)


root = tk.Tk()
root.title("PUF Extrapolator")
root.geometry("1000x800")

def_font = tk.font.nametofont("TkDefaultFont")
def_font.config(size=12)

frame = ttk.Frame(root, padding="10 10 10 10")
# frame.grid_columnconfigure(1, weight=1)
frame.grid(column=2, row=10)

# Ratio train/test
ratio_var = tk.DoubleVar()
ratio_var.set(0.3)

# Number of bootstrap iterations
iters_var = tk.IntVar()
```

```
iters_var.set(1000)

# Number of devices per bootstrap
sample_size_var = tk.IntVar()
sample_size_var.set(10)

# CRP file selected
file_var = tk.StringVar()
# file_var.set("/home/vinagres/Documents/PhD/data/sram/test_crps_ideal.txt")

file_label = tk.Label(frame, text="Ratio train/test")
file_label.grid(row=0, column=0, sticky=tk.W)

ratio_box = ttk.Spinbox(
    frame, from_=0.1, to=1.0, increment=0.1, textvariable=ratio_var, wrap=True
)
ratio_box.grid(row=0, column=1, sticky=tk.W)

iters_label = tk.Label(frame, text="Bootstrap iterations")
iters_label.grid(row=1, column=0, sticky=tk.W)

iters_box = ttk.Spinbox(
    frame, from_=100, to=10000, increment=100, textvariable=iters_var, wrap=True
)
iters_box.grid(row=1, column=1, sticky=tk.W)

sample_label = tk.Label(frame, text="Sample Size")
sample_label.grid(row=2, column=0, sticky=tk.W)

blocks_box = ttk.Spinbox(
    frame, from_=1, to=20, increment=1, textvariable=sample_size_var, wrap=True
)
blocks_box.grid(row=2, column=1, sticky=tk.W)

file_label = tk.Label(frame, textvariable=file_var)
file_label.grid(row=3, column=0, sticky=tk.W)

file_btn = tk.Button(frame, text="Select CRPs file", command=load_file)
file_btn.grid(row=4, column=0, sticky=tk.W, padx=0, pady=10)
model_btn = tk.Button(frame, text="Extrapolate", command=start_model)
model_btn.grid(row=4, column=0, sticky=tk.W, padx=150, pady=10)

pb = ttk.Progressbar(frame, orient="horizontal", mode="indeterminate",
 ↪  length=400)
pb.grid(row=5, padx=5, pady=5, sticky=tk.W)

# Results text
text = tk.Text(frame, font="monospace 14", width=80, height=15)
text.grid(row=6, column=0, columnspan=2, rowspan=1, sticky=tk.W)
text.configure(state="disabled")
text.bind("<1>", lambda _: text.focus_set())

tk.Label(frame, text=help, justify="left").grid(row=11, pady=10, sticky=tk.W)

root.mainloop()
```

Listing D.2: Python source code for the TK Graphical User Interface

Grid Search

IX.E

```
using Random
using Distributions
using Statistics
using CSV
using DataFrames
import Base.Threads

include("utils.jl")

const Metrics = Dict{Symbol,Union{Int,Float64}}

function gen_fdiff_mat(mu, sigma, D, C)
    Fdiff = Normal(mu, sigma)
    fdiff = Array{Float64}(undef, D, C)
    for c in 1:C
        fdiff[:,c] = rand(Fdiff, D)
    end
    return fdiff
end


function gen_crps(fdiff)
    D, C = size(fdiff)
    crps = Array{Int}(undef, D, C)
    for c in 1:C
        crps[:,c] = fdiff[:,c] .≥ 1
    end
    return crps
end

function handle_group(mu, sigma)
    challenges = trunc.(Int, range(4, 512; length=30))
    devices = trunc.(Int, range(4, 500; length=25))
    num_samples = length(challenges) * length(devices)
    results = Array{Metrics}(undef, num_samples)
    D_MAX = max(devices...)
    C_MAX = max(challenges...)

    fdiff = gen_fdiff_mat(mu, sigma, D_MAX, C_MAX)
    crps = gen_crps(fdiff)

    count::Int = 1
    @inbounds for D in devices
        @inbounds for C in challenges
            crps_group = view(crps, 1:D, 1:C)
            ba = mean(crps_group, dims=1)
            unif = mean(crps_group, dims=2)
            hba = shannon.(ba)
            local alpha, beta
            try
                beta_fit = fit_mle(Beta, ba)
                alpha = getfield(beta_fit, 1)
                beta = getfield(beta_fit, 2)
            catch
                alpha = 0
                beta = 0
```

```
            end
            results[count] =  Dict(
                :cv => sigma / mu, :z => mu / sigma,
                :C => C, :D => D,
                :alpha => alpha, :beta => beta,
                :fdiff_mean => mu,
                :fdiff_sigma => sigma,
                :ba_mean => mean(ba), :ba_var => var(ba),
                :hba_mean => mean(hba), :hba_var => var(hba),
                :unif_mean => mean(unif), :unif_var => var(unif)
                )
            count += 1
        end # challenges
    end # devices
    return results
end

function main()
    fdiff_means = range(-1e3, 1e3; length=51)
    fdiff_sigmas = range(0, 3e3; length=50)
    # cvs = range(1e-3, 20; length=51)
    # zetas = range(-20, 20; length=51)

    num_samples = length(fdiff_means) * length(fdiff_sigmas)
    results = Array{Array{Metrics}}(undef, num_samples)
    count::Int = 1
    Threads.@threads for mu in fdiff_means
        println("FDiff Mean $(mu)")
        @inbounds for sigma in fdiff_sigmas
            results[count] = handle_group(mu, sigma)
            count += 1
        end
    end
    return [results...]
end

function save_results(results, path::String)
    results_df = vcat(DataFrame.(results)...)
    CSV.write(path, results_df)
end

save_results(main(), "etl_fdiff_ba_beta_fit.csv")
```

Listing E.1: Julia implementation of the Grid Search to study the relationship between the frequency difference distribution and PUF metrics

```
using StatsBase
using Statistics
using Random
using Distributions
using DataFrames
using CSV
using Printf
import Base.Threads

# const Metrics = Dict{Symbol,Union{Int,Float64}}
const Metrics = Dict{Symbol,Float64}

const C_MAX::Int = 512
const D_MAX::Int = 100
```

```
15
16  hamming_weight(x::AbstractArray)::Float64 = sum(x) / length(x)
17  hamming_dist(x::AbstractArray, y::AbstractArray)::Float64 = hamming_weight(x
    ↪  .≠ y)
18
19  # Each column is a device
20  function bitaliasing(crps::AbstractArray{T, 2})::Array{Float64} where T <:
    ↪  Number
21      _, C = size(crps)
22      ba = Array{Float64}(undef, C)
23      @inbounds for i = 1:C
24          ba[i] = hamming_weight(@view crps[:, i])
25      end
26      ba
27  end
28
29  # Julia is column-major so it's better to computer per column
30  uniformity(crps) = bitaliasing(transpose(crps))
31
32  # Each column is a device
33  function uniqueness(crps)::Array{Float64}
34      _, D = size(crps)
35      num_pairs::Int = (D * (D - 1)) / 2
36      count::Int = 1
37      uniq = Array{Float64}(undef, num_pairs)
38      @inbounds for i = 1:D-1
39          for j = i+1:D
40              uniq[count] = hamming_dist(view(crps, :, i), view(crps, :, j))
41              count += 1
42          end
43      end
44      return uniq
45  end
46
47  """Generate a vector of random bits"""
48  function random_bits(ba::Array{Float64})::Array{Int,1}
49      thresh = rand(Float64, length(ba))
50      bits = Array{Int}(undef, length(ba))
51      # bits .*= t .< ba[i]
52      @inbounds for (i, t) in enumerate(thresh)
53          bits[i] = t ≤ ba[i] ? 1 : 0
54      end
55      return bits
56  end
57
58
59  function gen_crps(ba_mean::Float64, ba_var::Float64, D::Int=D_MAX,
    ↪  C::Int=C_MAX)::Array{Int8,2}
60      ba_dist = rand(Normal(ba_mean, sqrt(ba_var)), C)
61      ba = clamp.(ba_dist, 0, 1)
62      crps = Array{Int8}(undef, D, C)
63      @inbounds for d = 1:D
64          crps[d, :] = random_bits(ba)
65      end
66      return crps
67  end
68
69  function metrics(crps::AbstractArray)::Metrics
70      D, C = size(crps)
71      unif = uniformity(crps)
72      ba = bitaliasing(crps)
73      ba_sqrd = ba .* ba
```

```
    cov_ba_ba_sqrd = cov(ba, ba_sqrd)
    pdiff = @. 2 * ba * (1 - ba)
    peq = @. ba^2 + (1 - ba)^2
    uniq = uniqueness(transpose(crps))
    Dict(
        :D => D,
        :C => C,
        :unif_mean => mean(unif),
        :unif_var => var(unif),
        :ba_mean => mean(ba),
        :ba_var => var(ba),
        :ba_sqrd_mean => mean(ba_sqrd),
        :ba_sqrd_var => var(ba_sqrd),
        :cov_ba_ba_sqrd => cov_ba_ba_sqrd,
        :pdiff_mean => mean(pdiff),
        :pdiff_var => var(pdiff),
        :peq_mean => mean(peq),
        :peq_var => var(peq),
        :uniq_mean => mean(uniq),
        :uniq_var => var(uniq),
    )
end

function handle_case(ba_mean::Float64, ba_var::Float64, devices::Array{Int,1},
 ↪  challenges::Array{Int,1})::Array{Metrics}
    crps = gen_crps(ba_mean, ba_var)
    results = Array{Metrics}(undef, length(challenges) * length(devices))
    count::Int = 1
    @inbounds for D in devices
        @inbounds for C in challenges
            results[count] = metrics(@view crps[1:D, 1:C])
            count += 1
        end
    end
    return results
end

function grid_search()
    cd("/home/vinagres/Documents/PhD")

    ba_means = range(0, stop = 1, length = 10)
    ba_vars = range(0, stop = 0.1, length = 10)

    devices = trunc.(Int, range(2, stop = D_MAX, length = 350))
    challenges = trunc.(Int, range(2, stop = C_MAX, length = 50))

    Threads.@threads for ba_mean in ba_means
        count::Int = 1
        results = Array{Array{Metrics}}(undef, length(ba_vars))
        println(@sprintf "Mean %.2f" ba_mean)
        @inbounds for ba_var in ba_vars
            results[count] = handle_case(ba_mean, ba_var, challenges, devices)
            count += 1
        end
        path = @sprintf "data/metrics/search_space/search_space_%.02f.csv"
         ↪  ba_mean
        @show path
        save_results(results, path)
    end
    # vcat(results...)
end

```

```
grid_search()
```

Listing E.2: Julia implementation of the Grid Search to study the relationship between PUF metrics

```
using Distributions
using Random
using CSV
using DataFrames

function grid_search()
    Zetas = range(-20, 20, length=201)
    Sigmas = range(1, 100, length=50)
    results = Array{Dict{Symbol,Float64}}(undef, length(Zetas) * length(Sigmas))
    count = 1

    for z in Zetas, sigma in Sigmas
        mu = z * sigma
        F = rand(Normal(mu, sigma), 1024)
        Fabs = abs.(F)
        ttr = 1 ./ Fabs

        fit_fabs = fit_mle(Gamma, Fabs)
        fit_ttr_wald = fit_mle(InverseGaussian, ttr)
        fit_ttr_gamma = fit_mle(Gamma, ttr)

        results[count] = Dict(
            :mu => mu, :sigma => sigma, :z => z, :cv => 1 / z,
            :fabs_alpha => getfield(fit_fabs, 1), :fabs_beta =>
            ↪  getfield(fit_fabs, 2),
            :ttr_mean => getfield(fit_ttr_wald, 1), :ttr_lambda =>
            ↪  getfield(fit_ttr_wald, 2),
            :ttr_alpha => getfield(fit_ttr_gamma, 1), :ttr_beta =>
            ↪  getfield(fit_ttr_gamma, 2)
        )
        count += 1
    end

    results_df = vcat(DataFrame.(results)...)
    CSV.write("time_to_response.csv", results_df)
end

grid_search()
```

Listing E.3: Julia implementation of the Grid Search for the Time-To-Response analysis

Bibliography

[1] S. V. Gutiérrez, P. Inglese, G. Di Natale, and E.-I. Vatajelu, 'Open automation framework for complex parametric electrical simulations,' in *2023 26th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, 2023, pp. 132–135. DOI: 10.1109/DDECS57882.2023.10139409.

[2] S. Vinagrero Gutiérrez, G. Di Natale, and E.-I. Vatajelu, 'Python framework for modular and parametric spice netlists generation,' *Electronics*, vol. 12, no. 18, p. 3970, 2023.

[3] S. V. Gutierrez, H. Martin, E. I. Vatajelu, and G. Di Natale, 'Sram-puf: Platform for acquisition of sram-based pufs from micro-controllers,' in *University Booth-IEEE Design Automation and Test Conference in Europe (DATE 2021)*, 2021.

[4] S. Vinagrero, H. Martin, A. de Bignicourt, E.-I. Vatajelu, and G. Di Natale, 'SRAM-Based PUF Readouts,' *Scientific Data*, vol. 10, no. 1, p. 333, 2023, ISSN: 2052-4463. DOI: 10.1038/s41597-023-02225-9.

[5] S. Vinagrero Gutierrez, H. Martin Gonzalez, A. De Bignicourt, E. I. Vatajelu, and G. Di Natale, *SRAM-Based PUF Readouts*, version 1.0.0, Zenodo, Jan. 2023. DOI: 10.5281/zenodo.7529513.

[6] S. Vinagrero Gutierrez, G. Di Natale, and I. Vatajelu, 'On-Line Reliability Estimation of Ring Oscillator PUF,' in *IEEE European Test Symposium (ETS 2022)*, IEEE, Ed., Barcelona, Spain: IEEE, May 2022. DOI: 10.1109/ETS54262.2022.9810418. [Online]. Available: https://hal.archives-ouvertes.fr/hal-03767650.

[7] S. V. Gutierrez, G. Di Natale, and E.-I. Vatajelu, 'On-Line Method to Limit Unreliability and Bit-Aliasing in RO-PUF,' in *2023 IEEE 29th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, IEEE, 2023, pp. 1–6.

[8] V. Kulagin *et al.*, 'On the Relation Between Reliability and Entropy in Physical Unclonable Functions,' *IEEE Design & Test*, pp. 1–1, 2024. DOI: 10.1109/MDAT.2024.3425791.

[9] J. D. R. Buchanan *et al.*, 'Forgery: 'fingerprinting' documents and packaging,' *Nature*, vol. 436, 2005. DOI: 10.1038/436475a.

[10] S. Bhasin, J.-L. Danger, S. Guilley, X. T. Ngo, and L. Sauvage, 'Hardware trojan horses in cryptographic ip cores,' in *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*, IEEE, 2013, pp. 15–29.

[11] X. T. Ngo *et al.*, 'Hardware trojan detection by delay and electromagnetic measurements,' in *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2015, pp. 782–787.

[12] X. T. Ngo, J.-L. Danger, S. Guilley, Z. Najm, and O. Emery, 'Hardware property checker for run-time hardware trojan detection,' in *2015 European Conference on Circuit Theory and Design (ECCTD)*, IEEE, 2015, pp. 1–4.

[13] S. Guilley and R. Pacalet, 'Soc security: A war against side-channels,' *Annals of Telecommunications-annales des télécommunications*, 2004.

[14] S. Guilley, L. Sauvage, J.-L. Danger, N. Selmane, and R. Pacalet, 'Silicon-level solutions to counteract passive and active attacks,' in *2008 5th Workshop on Fault Diagnosis and Tolerance in Cryptography*, IEEE, 2008, pp. 3–17.

[15] J. Guajardo *et al.*, 'Anti-counterfeiting, key distribution, and key storage in an ambient world via physical unclonable functions,' *Inf. Syst. Front.*, vol. 11, 2009. DOI: 10.1007/s10796-008-9142-z.

[16] J. Bringer, H. Chabanne, and T. Icart, *On physical Obfuscation of Cryptographic Algorithms*. Berlin, Heidelberg: Springer, 2009.

[17] R. Pappu, B. Recht, J. Taylor, and N. Gershenfeld, 'Physical one-way functions,' *Science*, vol. 297, no. 5589, pp. 2026–2030, 2002.

[18] K. Lofstrom, W. R. Daasch, and D. Taylor, 'Ic identification circuit using device mismatch,' in *2000 IEEE International Solid-State Circuits Conference. Digest of Technical Papers (Cat. No. 00CH37056)*, IEEE, 2000, pp. 372–373.

[19] W. J. Herschel, *The origin of finger-printing*. H. Milford, Oxford University Press, 1916.

[20] D. Bauder, 'An anti-counterfeiting concept for currency systems,' *Sandia National Labs, Albuquerque, NM, Tech. Rep. PTK-11990*, 1983.

[21] *Counterfeit Deterrent Features for the Next-Generation Currency Design, Appendix E.* Washington, DC: The National Academic Press, 1993.

[22] K. M. Tolk, 'Reflective particle technology for identification of critical components,' Sandia National Labs., Albuquerque, NM (United States), Tech. Rep., 1992.

[23] S. Paulus, N. Pohlmann, H. Reimer, P. Tuyls, and B. Škorić, 'Physical unclonable functions for enhanced security of tokens and tags,' in *ISSE 2006—Securing Electronic Busines Processes: Highlights of the Information Security Solutions Europe 2006 Conference*, Springer, 2006, pp. 30–37.

[24] B. Gassend, *Physical Random Functions*. MA, USA: MIT, 2003.

[25] B. Gassend, D. Clarke, M. Van Dijk, and S. Devadas, 'Silicon physical random functions,' in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, 2002, pp. 148–160.

[26] B. Gassend, D. Clarke, M. Van Dijk, and S. Devadas, 'Delay-based circuit authentication and applications,' in *Proceedings of the 2003 ACM symposium on Applied computing*, 2003, pp. 294–301.

[27] J. W. Lee, D. Lim, B. Gassend, G. E. Suh, M. Van Dijk, and S. 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)*, IEEE, 2004, pp. 176–179.

[28] D. Lim, *Extracting Secret Keys from Integrated Circuits*. MA, USA: MIT, 2004.

[29] M. Majzoobi, F. Koushanfar, and M. Potkonjak, 'Testing techniques for hardware security,' in *2008 IEEE International Test Conference*, IEEE, 2008, pp. 1–10.

[30] U. Rührmair, J. Sölter, and F. Sehnke, 'On the foundations of physical unclonable functions,' *Cryptology ePrint Archive*, 2009.

[31] S. Vrijaldenhoven, *Acoustical Physical Uncloneable Functions*. the Netherlands: Technische Universiteit Eindhoven, 2005.

[32] P. Tuyls, G. J. Schrijen, B. Škorić, J. Geloven, N. Verhaegh, and R. Wolters, *Read-Proof Hardware from Protective Coatings*. New York, NY: Springer, 2006.

[33] B. Škorić, S. Maubach, T. Kevenaar, and P. Tuyls, 'Information-theoretic analysis of capacitive physical unclonable functions,' *J. Appl. Phys.*, vol. 100, 2006. DOI: 10.1063/1.2209532.

[34] E. Simpson and P. Schaumont, 'Offline hardware/software authentication for reconfigurable platforms,' in *International Workshop on Cryptographic Hardware and Embedded Systems*, Springer, 2006, pp. 311–323.

[35] J. Guajardo, S. S. Kumar, G.-J. Schrijen, and P. Tuyls, 'Fpga intrinsic pufs and their use for ip protection,' in *International workshop on cryptographic hardware and embedded systems*, Springer, 2007, pp. 189–195.

[36] G. Dejean and D. Kirovski, *RF-DNA: Radio-Frequency Certificates of Authenticity*. Berlin, Heidelberg: Springer, 2007.

[37] R. Helinski, D. Acharyya, and J. Plusquellic, *A Physical Unclonable Function Defined Using Power Distribution System Equivalent Resistance Variations*. New York, NY: ACM, 2009.

[38] E. Öztürk, G. Hammouri, and B. Sunar, *Towards Robust Low Cost Authentication for Pervasive Devices*. Washington, DC: IEEE Computer Society, 2008.

[39] D. E. Holcomb, W. P. Burleson, and K. Fu, 'Power-up sram state as an identifying fingerprint and source of true random numbers,' *IEEE Transactions on Computers*, vol. 58, no. 9, pp. 1198–1210, 2009. DOI: 10.1109/TC.2008.212.

[40] L. Bossuet, X. T. Ngo, Z. Cherif, and V. Fischer, 'A puf based on a transient effect ring oscillator and insensitive to locking phenomenon,' *IEEE Transactions on Emerging Topics in Computing*, vol. 2, no. 1, pp. 30–36, 2013.

[41] B. Škorić, P. Tuyls, and W. Ophey, 'Robust key extraction from physical uncloneable functions,' in *Applied Cryptography and Network Security: Third International Conference, ACNS 2005, New York, NY, USA, June 7-10, 2005. Proceedings 3*, Springer, 2005, pp. 407–422.

[42] S. Katzenbeisser, Ü. Kocabaş, V. Rožić, A.-R. Sadeghi, I. Verbauwhede, and C. Wachsmann, 'Pufs: Myth, fact or busted? a security evaluation of physically unclonable functions (pufs) cast in silicon,' in *International Workshop on Cryptographic Hardware and Embedded Systems*, Springer, 2012, pp. 283–301.

[43] R. Maes, V. Rozic, I. Verbauwhede, P. Koeberl, E. van der Sluis, and V. van der Leest, 'Experimental evaluation of physically unclonable functions in 65 nm cmos,' in *2012 Proceedings of the ESSCIRC (ESSCIRC)*, 2012, pp. 486–489. DOI: 10.1109/ESSCIRC.2012.6341361.

[44] T. McGrath, I. E. Bagci, Z. M. Wang, U. Roedig, and R. J. Young, 'A puf taxonomy,' *Applied physics reviews*, vol. 6, no. 1, 2019.

[45] Y. Cao, J. Xu, J. Wu, S. Wu, Z. Huang, and K. Zhang, 'Advances in physical unclonable functions based on new technologies: A comprehensive review,' *Mathematics*, vol. 12, no. 1, p. 77, 2023.

[46] U. Rührmair, H. Busch, and S. Katzenbeisser, 'Strong pufs: Models, constructions, and security proofs,' *Towards Hardware-Intrinsic Security: Foundations and Practice*, pp. 79–96, 2010.

[47] R. Maes and I. Verbauwhede, 'Physically unclonable functions: A study on the state of the art and future research directions,' *Towards Hardware-Intrinsic Security*, pp. 3–37, 2010.

[48] J. Zhang and L. Wan, 'Cmos: Dynamic multi-key obfuscation structure for strong pufs,' *ArXiv*, vol. abs/1806.02011, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:195347140.

[49] S. R. Sahoo, S. Kumar, and K. Mahapatra, 'A novel configurable ring oscillator puf with improved reliability using reduced supply voltage,' *Microprocessors and Microsystems*, vol. 60, pp. 40–52, 2018.

[50] Z. Cherif, J.-L. Danger, S. Guilley, and L. Bossuet, 'An easy-to-design puf based on a single oscillator: The loop puf,' in *2012 15th Euromicro Conference on Digital System Design*, IEEE, 2012, pp. 156–162.

[51] C.-E. Yin, G. Qu, and Q. Zhou, 'Design and implementation of a group-based ro puf,' in *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2013, pp. 416–421.

[52] D. E. Holcomb, W. P. Burleson, K. Fu, *et al.*, 'Initial sram state as a fingerprint and source of true random numbers for rfid tags,' in *Proceedings of the Conference on RFID Security*, vol. 7, 2007, p. 01.

[53] J. S. Kim, M. Patel, H. Hassan, and O. Mutlu, 'The dram latency puf: Quickly evaluating physical unclonable functions by exploiting the latency-reliability tradeoff in modern commodity dram devices,' in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, IEEE, 2018, pp. 194–207.

[54] M. Majzoobi, F. Koushanfar, and S. Devadas, 'Fpga puf using programmable delay lines,' in *2010 IEEE international workshop on information forensics and security*, IEEE, 2010, pp. 1–6.

[55] J. B. Wendt and M. Potkonjak, 'Hardware obfuscation using puf-based logic,' in *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, IEEE, 2014, pp. 270–271.

[56] R. Della Sala, D. Bellizia, and G. Scotti, 'A novel ultra-compact fpga puf: The dd-puf,' *Cryptography*, vol. 5, no. 3, p. 23, 2021.

[57] R. Della Sala, D. Bellizia, and G. Scotti, 'A lightweight fpga compatible weak-puf primitive based on xor gates,' *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 6, pp. 2972–2976, 2022.

[58] P. Koeberl, Ü. Kocabaş, and A.-R. Sadeghi, 'Memristor pufs: A new generation of memory-based physically unclonable functions,' in *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2013, pp. 428–431.

[59] G. S. Rose, N. McDonald, L.-K. Yan, and B. Wysocki, 'A write-time based memristive puf for hardware security applications,' in *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, IEEE, 2013, pp. 830–833.

[60] P.-Y. Chen, R. Fang, R. Liu, C. Chakrabarti, Y. Cao, and S. Yu, 'Exploiting resistive cross-point array for compact design of physical unclonable function,' in *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, IEEE, 2015, pp. 26–31.

[61] Y. Gao, D. C. Ranasinghe, S. F. Al-Sarawi, O. Kavehei, and D. Abbott, 'Mrpuf: A novel memristive device based physical unclonable function,' in *International Conference on Applied Cryptography and Network Security*, Springer, 2015, pp. 595–615.

[62] E. I. Vatajelu, G. D. Natale, M. Barbareschi, L. Torres, M. Indaco, and P. Prinetto, 'Stt-mram-based puf architecture exploiting magnetic tunnel junction fabrication-induced variability,' *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 13, no. 1, pp. 1–21, 2016.

[63] S. C. Konigsmark, L. K. Hwang, D. Chen, and M. D. Wong, 'Cnpuf: A carbon nanotube-based physically unclonable function for secure low-energy hardware design,' in *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, IEEE, 2014, pp. 73–78.

[64] B. C. Grubel *et al.*, 'Silicon photonic physical unclonable function,' *Optics Express*, vol. 25, no. 11, pp. 12 710–12 721, 2017.

[65] V. C. Immler, 'Higher-order alphabet physical unclonable functions,' Ph.D. dissertation, Technische Universität München, 2019.

[66] C. Frisch and M. Pehl, 'Beware of the bias-statistical performance evaluation of higher-order alphabet pufs,' in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2022, pp. 1005–1010.

[67] M. Mustapa and M. Niamat, 'Relationship between number of stages in ropuf and crp generation on fpga,' in *Proceedings of the International Conference on Security and Management (SAM)*, The Steering Committee of The World Congress in Computer Science, Computer ..., 2014, p. 1.

[68] Y. Su, J. Holleman, and B. Otis, *A 1.6pj/bit 96% Stable Chip-ID Generating Circuit Using Process Variations*. Washington, DC: IEEE Computer Society, 2007.

[69] B. H. Calhoun and A. P. Chandrakasan, 'Static noise margin variation for sub-threshold sram in 65-nm cmos,' *IEEE Journal of solid-state circuits*, vol. 41, no. 7, pp. 1673–1679, 2006.

[70] K. Agarwal and S. Nassif, 'Statistical analysis of sram cell stability,' in *Proceedings of the 43rd annual design automation conference*, 2006, pp. 57–62.

[71] A. Alheyasat, G. Torrens, S. A. Bota, and B. Alorda, 'Estimation during design phases of suitable sram cells for puf applications using separatrix and mismatch metrics,' *Electronics*, vol. 10, no. 12, p. 1479, 2021.

[72] M. I. Baturone Castillo, M. Á. Prada Delgado, and S. Eiroa, 'Improved generation of identifiers, secret keys, and random numbers from srams,' *IEEE Transactions on Information Forensics and Security, 10 (12), 2653-2658.*, 2015.

[73] I. Karageorgos, M. M. Isgenc, S. Pagliarini, and L. Pileggi, 'Chip-to-chip authentication method based on sram puf and public key cryptography,' *Journal of Hardware and Systems Security*, vol. 3, pp. 382–396, 2019. [Online]. Available: https://github.com/ecelab-org/Split-Chip_authentication/tree/master.

[74] S. Kumar, J. Guajardo, R. Maes, G. J. Schrijen, and P. Tuyls, *Extended Abstract: The Butterfly PUF Protecting IP on Every FPGA*. Anaheim, CA, USA: IEEE International Workshop on Hardware-Oriented Security and Trust, 2008, HOST 2008, 2008.

[75] K. Xiao, M. T. Rahman, D. Forte, Y. Huang, M. Su, and M. Tehranipoor, 'Bit selection algorithm suitable for high-volume production of sram-puf,' in *2014 IEEE international symposium on hardware-oriented security and trust (HOST)*, IEEE, 2014, pp. 101–106.

[76] D. Kinniment and E. Chester, 'Design of an on-chip random number generator using metastability,' in *Proceedings of the 28th European Solid-State Circuits Conference*, IEEE, 2002, pp. 595–598.

[77] A. Alheyasat, G. Torrens, S. Bota, and B. Alorda, 'Selection of SRAM cells to improve reliable PUF implementation using cell mismatch metric,' in *2020 XXXV Conference on Design of Circuits and Integrated Systems (DCIS)*, IEEE, 2020, pp. 1–6.

[78] W. Wang, A. Singh, U. Guin, and A. Chatterjee, 'Exploiting power supply ramp rate for calibrating cell strength in sram pufs,' in *2018 IEEE 19th Latin-American Test Symposium (LATS)*, IEEE, 2018, pp. 1–6.

[79] E. I. Vatajelu, G. Di Natale, and P. Prinetto, 'Towards a highly reliable sram-based pufs,' in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2016, pp. 273–276.

[80] G. Torrens, A. Alheyasat, B. Alorda, and S. A. Bota, 'Sram-based puf reliability prediction using cell-imbalance characterization in the state space diagram,' *Electronics*, vol. 11, no. 1, p. 135, 2022.

[81] I. Baturone, M. A. Prada-Delgado, and S. Eiroa, 'Improved generation of identifiers, secret keys, and random numbers from srams,' *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 12, pp. 2653–2668, 2015.

[82] J. Lee, D.-W. Jee, and D. Jeon, 'Power-up control techniques for reliable sram puf,' *IEICE Electronics Express*, vol. 16, no. 13, pp. 20 190 296–20 190 296, 2019.

[83] M. Cortez, S. Hamdioui, A. Kaichouhi, V. van der Leest, R. Maes, and G.-J. Schrijen, 'Intelligent voltage ramp-up time adaptation for temperature noise reduction on memory-based puf systems,' *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 7, pp. 1162–1175, 2015.

[84] S. Pandey, S. Deyati, A. Singh, and A. Chatterjee, 'Noise-resilient sram physically unclonable function design for security,' in *2016 IEEE 25th Asian Test Symposium (ATS)*, 2016, pp. 55–60. DOI: 10.1109/ATS.2016.65.

[85] D. Mukhopadhyay and R. S. Chakraborty, *Hardware security: design, threats, and safeguards*. CRC Press, 2014.

[86] U. Rührmair, U. Schlichtmann, and W. Burleson, 'Special session: How secure are pufs really? on the reach and limits of recent puf attacks,' in *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2014, pp. 1–4. DOI: 10.7873/DATE.2014.359.

[87] T. Kroeger, W. Cheng, S. Guilley, J.-L. Danger, and N. Karimi, 'Cross-puf attacks on arbiter-pufs through their power side-channel,' in *2020 IEEE International Test Conference (ITC)*, IEEE, 2020, pp. 1–5.

[88] J. Mahmod and M. Hicks, 'Sram has no chill: Exploiting power domain separation to steal on-chip secrets,' in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022, pp. 1043–1055.

[89] X. T. Ngo *et al.*, 'Cryptographically secure shield for security ips protection,' *IEEE Transactions on Computers*, vol. 66, no. 2, pp. 354–360, 2016.

[90] J.-M. Cioranesco *et al.*, 'Cryptographically secure shields,' in *2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, IEEE, 2014, pp. 25–31.

[91] L. Tebelmann, J.-L. Danger, and M. Pehl, 'Self-secured puf: Protecting the loop puf by masking,' in *International Workshop on Constructive Side-Channel Analysis and Secure Design*, Springer, 2020, pp. 293–314.

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

[93] D. Nedospasov, J.-P. Seifert, C. Helfmeier, and C. Boit, 'Invasive puf analysis,' in *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*, IEEE, 2013, pp. 30–38.

[94] R. Maes, P. Tuyls, and I. Verbauwhede, *Statistical Analysis of Silicon PUF Responses for Device Identification*. Germany: Berlin, 2008.

[95] A. Maiti and P. Schaumont, 'The impact of aging on a physical unclonable function,' *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 9, pp. 1854–1864, 2013.

[96] D. Ganta and L. Nazhandali, 'Study of ic aging on ring oscillator physical unclonable functions,' in *Fifteenth international symposium on quality electronic design*, IEEE, 2014, pp. 461–466.

[97] N. Karimi, J.-L. Danger, F. Lozac'h, and S. Guilley, 'Predictive aging of reliability of two delay pufs,' in *Security, Privacy, and Applied Cryptography Engineering: 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings 6*, Springer, 2016, pp. 213–232.

[98] M. Barbareschi, G. D. Natale, L. Torres, and A. Mazzeo, 'A ring oscillator-based identification mechanism immune to aging and external working conditions,' *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. PP, pp. 1–23, Aug. 2017. DOI: 10.1109/TCSI.2017.2727546.

[99] P. Solé, W. Cheng, S. Guilley, and O. Rioul, 'Bent sequences over hadamard codes for physically unclonable functions,' in *2021 IEEE International Symposium on Information Theory (ISIT)*, IEEE, 2021, pp. 801–806.

[100] M.-D. Yu and S. Devadas, 'Secure and robust error correction for physical unclonable functions,' *IEEE Design & Test of Computers*, vol. 27, no. 1, pp. 48–65, 2010.

[101] M. S. Kirkpatrick and E. Bertino, *Software Techniques to Combat Drift in PUF-Based Authentication Systems*. Cologne, Germany: Workshop on Secure Component and System Identification (SECSI 2010), 2010.

[102] J. Béguinot, W. Cheng, J.-L. Danger, S. Guilley, O. Rioul, and V. Yli-Mäyry, 'Reliability of ring oscillator pufs with reduced helper data,' in *International Workshop on Security*, Springer, 2023, pp. 36–56.

[103] J. Delvaux, D. Gu, D. Schellekens, and I. Verbauwhede, 'Helper data algorithms for puf-based key generation: Overview and analysis,' *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 6, pp. 889–902, 2014.

[104] J. Delvaux and I. Verbauwhede, 'Key-recovery attacks on various ro puf constructions via helper data manipulation,' in *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2014, pp. 1–6.

[105] Z. Cherif, J.-L. Danger, F. Lozac'h, Y. Mathieu, and L. Bossuet, 'Evaluation of delay pufs on cmos 65 nm technology: Asic vs fpga,' in *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, 2013, pp. 1–8.

[106] D. Merli, F. Stumpf, and C. Eckert, 'Improving the quality of ring oscillator pufs on fpgas,' in *Proceedings of the 5th workshop on embedded systems security*, 2010, pp. 1–9.

[107] R. Hesselbarth, F. Wilde, C. Gu, and N. Hanley, 'Large scale ro puf analysis over slice type, evaluation time and temperature on 28nm xilinx fpgas,' in *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, IEEE, 2018, pp. 126–133.

[108] N. Metropolis and S. Ulam, 'The monte carlo method,' *Journal of the American statistical association*, vol. 44, no. 247, pp. 335–341, 1949.

[109] L. W. Nagel and D. Pederson, 'SPICE (simulation program with integrated circuit emphasis),' EECS Department, University of California, Berkeley, Tech. Rep. UCB/ERL M382, Apr. 1973. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/1973/22871.html.

[110] F. Salvaire, *Pyspice*. [Online]. Available: https://pyspice.fabrice-salvaire.fr.

[111] A. Xynos and V. Tenentes, 'Metaspice: Metaprogramming spice framework for the design space exploration of puf circuits,' in *2023 12th International Conference on Modern Circuits and Systems Technologies (MOCAST)*, 2023, pp. 1–4. DOI: 10.1109/MOCAST57943.2023.10176643.

[112] J. Bachrach *et al.*, 'Chisel: Constructing hardware in a scala embedded language,' in *Proceedings of the 49th Annual Design Automation Conference*, 2012, pp. 1216–1225.

[113] S. Jiang, P. Pan, Y. Ou, and C. Batten, 'Pymtl3: A python framework for open-source hardware modeling, generation, simulation, and verification,' *IEEE Micro*, vol. 40, no. 4, pp. 58–66, 2020.

[114] D. Batas and H. Fiedler, 'A python interface for spice-based simulations,' in *ICSES 2010 International Conference on Signals and Electronic Circuits*, 2010, pp. 161–164.

[115] J. K. Ousterhout, G. T. Hamachi, R. N. Mayo, W. S. Scott, and G. S. Taylor, 'The magic vlsi layout system,' *IEEE Design & Test of Computers*, vol. 2, no. 1, pp. 19–30, 1985.

[116] A. A. Nielsen *et al.*, 'Genetic circuit design automation,' *Science*, vol. 352, no. 6281, aac7341, 2016.

[117] S. Jain and H. C. Gea, 'Pcb layout design using a genetic algorithm,' 1996.

[118] C. D. Chapman, K. Saitou, and M. J. Jakiela, 'Genetic algorithms as an approach to configuration and topology design,' 1994.

[119] K. Shahookar and P. Mazumder, 'A genetic approach to standard cell placement using meta-genetic parameter optimization,' *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 9, no. 5, pp. 500–511, 1990.

[120] R. S. Zebulum, M. A. Pacheco, and M. M. B. Vellasco, *Evolutionary electronics: automatic design of electronic circuits and systems by genetic algorithms*. CRC press, 2018.

[121] N. Lourenço, R. Martins, A. Canelas, R. Povoa, and N. Horta, 'Aida: Layout-aware analog circuit-level sizing with in-loop layout generation,' *Integration*, vol. 55, pp. 316–329, 2016.

[122] M. R. Guthaus, J. E. Stine, S. Ataei, B. Chen, B. Wu, and M. Sarwar, 'Openram: An open-source memory compiler,' in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, IEEE, 2016, pp. 1–6.

[123] B. Naveen and K. Raghunathan, 'An automatic netlist-to-schematic generator,' *IEEE Design & Test of Computers*, vol. 10, no. 1, pp. 36–41, 1993.

[124] S. Youssef, F. Javid, D. Dupuis, R. Iskander, and M.-M. Louerat, 'A python-based layout-aware analog design methodology for nanometric technologies,' in *2011 IEEE 6th International Design and Test Workshop (IDT)*, IEEE, 2011, pp. 62–67.

[125] T. Casper, D. Duque, S. Schöps, and H. De Gersem, 'Automated netlist generation for 3d electrothermal and electromagnetic field problems,' *Journal of Computational Electronics*, vol. 18, no. 4, pp. 1306–1332, 2019.

[126] Y.-S. Jehng, L.-G. Chen, and T.-M. Parng, 'ASG: Automatic schematic generator,' *Integration*, vol. 11, no. 1, pp. 11–27, 1991.

[127] G. Huang *et al.*, 'Machine learning for electronic design automation: A survey,' *ACM Trans. Des. Autom. Electron. Syst.*, vol. 26, no. 5, Jun. 2021, ISSN: 1084-4309. DOI: 10.1145/3451179.

[128] M. Cortez, A. Dargar, S. Hamdioui, and G.-J. Schrijen, 'Modeling sram start-up behavior for physical unclonable functions,' in *2012 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, IEEE, 2012, pp. 1–6.

[129] M. Barbareschi, E. Battista, A. Mazzeo, and N. Mazzocca, 'Testing 90 nm microcontroller sram puf quality,' in *2015 10th International Conference on Design Technology of Integrated Systems in Nanoscale Era (DTIS)*, 2015, pp. 1–6. DOI: 10.1109/DTIS.2015.7127360.

[130] F. Wilde, 'Large scale characterization of sram on infineon xmc microcontrollers as puf,' in *Proceedings of the Fourth Workshop on Cryptography and Security in Computing Systems*, ser. CS2 '17, Stockholm, Sweden: Association for Computing Machinery, 2017, pp. 13–18, ISBN: 9781450348690. DOI: 10.1145/3031836.3031839.

[131] R. Wang, G. Selimis, R. Maes, and S. Goossens, 'Long-term continuous assessment of sram puf and source of random numbers,' in *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2020, pp. 7–12. DOI: 10.23919/DATE48585.2020.9116353.

[132] R. Maes and V. Van Der Leest, 'Countering the effects of silicon aging on sram pufs,' in *2014 IEEE International symposium on hardware-oriented security and trust (HOST)*, IEEE, 2014, pp. 148–153.

[133] A. Garg and T. T. Kim, 'Design of sram puf with improved uniformity and reliability utilizing device aging effect,' in *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, IEEE, 2014, pp. 1941–1944.

[134] A. Roelke and M. R. Stan, 'Controlling the reliability of sram pufs with directed nbti aging and recovery,' *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 10, pp. 2016–2026, 2018.

[135] P. Saraza-Canflanca *et al.*, 'Improving the reliability of sram-based pufs under varying operation conditions and aging degradation,' *Microelectronics Reliability*, vol. 118, p. 114 049, 2021.

[136] Y. Oren, A.-R. Sadeghi, and C. Wachsmann, 'On the effectiveness of the remanence decay side-channel to clone memory-based pufs,' in *Cryptographic Hardware and Embedded Systems - CHES 2013*, G. Bertoni and J.-S. Coron, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 107–125, ISBN: 978-3-642-40349-1.

[137] M. Liu, C. Zhou, Q. Tang, K. K. Parhi, and C. H. Kim, 'A data remanence based approach to generate 100% stable keys from an sram physical unclonable function,' in *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, IEEE, 2017, pp. 1–6.

[138] C. M. Mezzomo, A. Bajolet, A. Cathignol, R. Di Frenza, and G. Ghibaudo, 'Characterization and modeling of transistor variability in advanced cmos technologies,' *IEEE Transactions on Electron Devices*, vol. 58, no. 8, pp. 2235–2248, 2011.

[139] R. Sarpeshkar, T. Delbruck, and C. A. Mead, 'White noise in mos transistors and resistors,' *IEEE Circuits and Devices Magazine*, vol. 9, no. 6, pp. 23–29, 1993.

[140] C. G. Theodorou, M. Fadlallah, X. Garros, C. Dimitriadis, and G. Ghibaudo, 'Noise-induced dynamic variability in nano-scale cmos sram cells,' in *2016 46th European Solid-State Device Research Conference (ESSDERC)*, IEEE, 2016, pp. 256–259.

[141] C. G. Theodorou, E. G. Ioannidis, S. Haendler, C. A. Dimitriadis, and G. Ghibaudo, 'Dynamic variability in 14 nm fd-soi mosfets and transient simulation methodology,' *Solid-State Electronics*, vol. 111, pp. 100–103, 2015.

[142] A. Maiti, J. Casarona, L. McHale, and P. Schaumont, 'A large scale characterization of ro-puf,' in *2010 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, IEEE, 2010, pp. 94–99.

[143] A. Maiti, V. Gunreddy, and P. Schaumont, 'A systematic method to evaluate and compare the performance of physical unclonable functions,' *Embedded systems design with FPGAs*, pp. 245–267, 2013.

[144] E. Arikan, 'An inequality on guessing and its application to sequential decoding,' *IEEE Transactions on Information Theory*, vol. 42, no. 1, pp. 99–105, 1996.

[145] W.-C. Wang, Y. Yona, S. N. Diggavi, and P. Gupta, 'Design and analysis of stability-guaranteed pufs,' *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 4, pp. 978–992, 2017.

[146] Y. Hori, T. Yoshida, T. Katashita, and A. Satoh, 'Quantitative and statistical performance evaluation of arbiter physical unclonable functions on fpgas,' in *2010 International conference on reconfigurable computing and FPGAs*, IEEE, 2010, pp. 298–303.

[147] Z. C. Jouini, J.-L. Danger, and L. Bossuet, 'Performance evaluation of physically unclonable function by delay statistics,' in *2011 IEEE 9th International New Circuits and systems conference*, IEEE, 2011, pp. 482–485.

[148] M. I. O'CONNOR, M. I. VERBAUWHEDE, M. B. ROUZEYRE, and M. V. FISCHER, 'Modelisation et Caracterisation des Fonctions non Clonables Physiquement,' Ph.D. dissertation, Orange Labs, 2014.

[149] F. Armknecht, R. Maes, A.-R. Sadeghi, F.-X. Standaert, and C. Wachsmann, 'A formalization of the security features of physical functions,' in *2011 IEEE Symposium on Security and Privacy*, 2011, pp. 397–412. DOI: 10.1109/SP.2011.10.

[150] M. van Dijk and C. Jin, 'A theoretical framework for the analysis of physical unclonable function interfaces and its relation to the random oracle model,' *Cryptology ePrint Archive*, 2022.

[151] Y. Dodis, R. Ostrovsky, L. Reyzin, and A. Smith, 'Fuzzy extractors: How to generate strong keys from biometrics and other noisy data,' *SIAM J. Comput.*, vol. 38, 2008. DOI: 10.1137/060651380.

[152] J. Delvaux, D. Gu, I. Verbauwhede, M. Hiller, and M.-D. Yu, 'Efficient fuzzy extraction of puf-induced secrets: Theory and applications,' in *International Conference on Cryptographic Hardware and Embedded Systems*, Springer, 2016, pp. 412–431.

[153] K. Fukushima *et al.*, 'Delay puf assessment method based on side-channel and modeling analyzes: The final piece of all-in-one assessment methodology,' in *2016 IEEE Trustcom/BigDataSE/ISPA*, IEEE, 2016, pp. 201–207.

[154] A. Schaub, J.-L. Danger, S. Guilley, and O. Rioul, 'An improved analysis of reliability and entropy for delay pufs,' in *2018 21st Euromicro Conference on Digital System Design (DSD)*, 2018, pp. 553–560. DOI: 10.1109/DSD.2018.00096.

[155] F. Wilde and M. Pehl, 'On the confidence in bit-alias measurement of physical unclonable functions,' in *2019 17th IEEE International New Circuits and Systems Conference (NEWCAS)*, IEEE, 2019, pp. 1–4.

[156] M. Pehl, A. R. Punnakkal, M. Hiller, and H. Graeb, 'Advanced performance metrics for physical unclonable functions,' in *2014 International Symposium on Integrated Circuits (ISIC)*, IEEE, 2014, pp. 136–139.

[157] F. Wilde, B. M. Gammel, and M. Pehl, 'Spatial correlation analysis on physical unclonable functions,' *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 6, pp. 1468–1480, 2018. DOI: 10.1109/TIFS.2018.2791341.

[158] N. Bruneau *et al.*, 'Development of the unified security requirements of pufs during the standardization process,' in *Innovative Security Solutions for Information Technology and Communications: 11th International Conference, SecITC 2018, Bucharest, Romania, November 8–9, 2018, Revised Selected Papers 11*, Springer, 2019, pp. 314–330.

[159] Y. Wei, W. Rao, and N. Devroye, 'Apuf production line faults: Uniqueness and testing,' in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2023, pp. 1–6.

[160] F. K. Wilde, 'Metrics for physical unclonable functions,' Ph.D. dissertation, Universität München, 2021.

[161] F. Ganji, S. Tajik, F. Fäßler, and J.-P. Seifert, 'Having no mathematical model may not secure pufs,' *Journal of Cryptographic Engineering*, vol. 7, pp. 113–128, 2017.

[162] T. Arul, N. A. Anagnostopoulos, S. Reißig, and S. Katzenbeisser, 'A study of the spatial auto-correlation of memory-based physical unclonable functions,' in *2020 European Conference on Circuit Theory and Design (ECCTD)*, IEEE, 2020, pp. 1–4.

[163] N. Mexis *et al.*, 'Spatial correlation in weak physical unclonable functions: A comprehensive overview,' in *2023 26th Euromicro Conference on Digital System Design (DSD)*, IEEE, 2023, pp. 70–78.

[164] C.-E. Yin and G. Qu, 'Improving puf security with regression-based distiller,' in *Proceedings of the 50th Annual Design Automation Conference*, 2013, pp. 1–6.

[165] C.-E. Yin and G. Qu, 'Obtaining statistically random information from silicon physical unclonable functions,' *IEEE Transactions on Emerging Topics in Computing*, vol. 2, no. 2, pp. 96–106, 2014.

[166] B. M. S. Bahar Talukder, F. Ferdaus, and M. T. Rahman, 'Memory-based pufs are vulnerable as well: A non-invasive attack against sram pufs,' *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 4035–4049, 2021. DOI: 10.1109/TIFS.2021.3101045.

[167] C. Utz, J. Tobisch, and G. T. Becker, 'Analysis of 1000 arbiter puf based rfid tags,' 2016.

[168] M. C. Borja and J. Haigh, 'The birthday problem,' *Significance*, vol. 4, no. 3, pp. 124–127, 2007.

[169] K. Suzuki, D. Tonien, K. Kurosawa, and K. Toyota, 'Birthday paradox for multi-collisions,' in *Information Security and Cryptology–ICISC 2006: 9th International Conference, Busan, Korea, November 30-December 1, 2006. Proceedings 9*, Springer, 2006, pp. 29–40.

[170] C. Gu, W. Liu, N. Hanley, R. Hesselbarth, and M. O'Neill, 'A theoretical model to link uniqueness and min-entropy for puf evaluations,' *IEEE Transactions on Computers*, vol. 68, no. 2, pp. 287–293, 2018.

[171] L. Feiten, M. Sauer, and B. Becker, 'On metrics to quantify the inter-device uniqueness of pufs,' *Cryptology ePrint Archive*, 2016.

[172] R. Maes, 'An accurate probabilistic reliability model for silicon pufs,' in *International Conference on Cryptographic Hardware and Embedded Systems*, Springer, 2013, pp. 73–89.

[173] J.-L. Danger, S. Guilley, and A. Schaub, 'Two-metric helper data for highly robust and secure delay pufs,' in *2019 IEEE 8th International Workshop on Advances in Sensors and Interfaces (IWASI)*, IEEE, 2019, pp. 184–188.

[174] V. Immler, M. Hiller, J. Obermaier, and G. Sigl, 'Take a moment and have some t: Hypothesis testing on raw puf data,' in *2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, IEEE, 2017, pp. 128–129.

[175] D. Chatterjee, A. Hazra, and D. Mukhopadhyay, 'Formal analysis of puf instances leveraging correlation-spectra in boolean functions,' in *International Conference on Security, Privacy, and Applied Cryptography Engineering*, Springer, 2019, pp. 142–158.

[176] B. Willsch, J. Hauser, S. Dreiner, A. Goehlich, and H. Vogt, 'Statistical tests to determine spatial correlations in the response behavior of puf,' in *2016 12th Conference on Ph. D. Research in Microelectronics and Electronics (PRIME)*, IEEE, 2016, pp. 1–4.

[177] C.-E. D. Yin and G. Qu, 'Lisa: Maximizing ro puf's secret extraction,' in *2010 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, IEEE, 2010, pp. 100–105.

[178] J.-L. Danger, S. Guilley, P. Nguyen, and O. Rioul, 'Pufs: Standardization and evaluation,' in *2016 Mobile System Technologies Workshop (MST)*, IEEE, 2016, pp. 12–18.

[179] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, and E. Barker, 'A statistical test suite for random and pseudorandom number generators for cryptographic applications,' Booz-allen and hamilton inc mclean va, Tech. Rep., 2001.

[180] M. Shiozaki, Y. Hori, and T. Fujino, 'Entropy estimation of physically unclonable functions with offset error,' *Cryptology ePrint Archive*, 2020.

[181] A. A. Pour *et al.*, 'Puf enrollment and life cycle management: Solutions and perspectives for the test community,' in *2020 IEEE European Test Symposium (ETS)*, 2020, pp. 1–10. DOI: 10.1109/ETS48528.2020.9131578.

[182] R. Valles-Novo, A. Martinez-Sanchez, and W. Che, 'Boosting entropy and enhancing reliability for physically unclonable functions,' in *2020 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, IEEE, 2020, pp. 1–6.

[183] H.-P. Nguyen, T.-N. Nguyen, Y.-S. Seo, D. Hwang, and D. Shin, 'Correction of bit-aliasing in memristor-based physically unclonable functions with timing variability,' *IEEE Access*, vol. 7, pp. 135 312–135 321, 2019.

[184] A. Garg, Z. C. Lee, L. Lu, and T. T.-H. Kim, 'Improving uniformity and reliability of sram pufs utilizing device aging phenomenon for unique identifier generation,' *Microelectronics Journal*, vol. 90, pp. 29–38, 2019.

[185] A. Schaub, J.-L. Danger, S. Guilley, and O. Rioul, 'Reliability and entropy of delay pufs: A theoretical analysis,' in *16th International Workshop on Cryptographic Architectures Embedded in Logic Devices (CryptArchi 2018)*, 2018.

[186] H. Martin, E.-I. Vatajelu, G. Di Natale, and O. Keren, 'On the Reliability of the Ring Oscillator Physically Unclonable Functions,' in *2019 IEEE 4th International Verification and Security Workshop (IVSW)*, 2019, pp. 25–30. DOI: 10.1109/IVSW.2019.8854401.

[187] M. Bhargava and K. Mai, 'An efficient reliable puf-based cryptographic key generator in 65nm cmos,' in *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2014, pp. 1–6.

[188] A. Schaub, J.-L. Danger, O. Rioul, and S. Guilley, 'The big picture of delay-PUF dependability,' in *2020 European Conference on Circuit Theory and Design (ECCTD)*, IEEE, 2020, pp. 1–4.

[189] R. Maes, P. Tuyls, and I. Verbauwhede, *Low-Overhead Implementation of a Soft Decision Helper Data Algorithm for SRAM PUFs*. Berlin, Heidelberg: Springer, 2009.

[190] R. Maes, A. Van Herrewege, and I. Verbauwhede, 'Pufky: A fully functional puf-based cryptographic key generator,' in *International Workshop on Cryptographic Hardware and Embedded Systems*, Springer, 2012, pp. 302–319.

[191] M. Bhargava, C. Cakir, and K. Mai, 'Reliability enhancement of bi-stable pufs in 65nm bulk cmos,' in *2012 IEEE International Symposium on Hardware-Oriented Security and Trust*, IEEE, 2012, pp. 25–30.

[192] R. Cao, N. Mei, and Q. Lian, 'Method for improving the reliability of sram-based puf using convolution operation,' *Electronics*, vol. 11, no. 21, p. 3493, 2022.

[193] M. T. Rahman, D. Forte, F. Rahman, and M. Tehranipoor, 'A pair selection algorithm for robust ro-puf against environmental variations and aging,' in *2015 33rd IEEE International Conference on Computer Design (ICCD)*, IEEE, 2015, pp. 415–418.

[194] S. Masoumian *et al.*, 'Modeling and analysis of sram puf bias patterns in 14nm and 7nm finfet technology nodes,' in *2023 IFIP/IEEE 31st International Conference on Very Large Scale Integration (VLSI-SoC)*, IEEE, 2023, pp. 1–6.

[195] Z. U. Abideen, R. Wang, T. D. Perez, G.-J. Schrijen, and S. Pagliarini, 'Impact of orientation on the bias of sram-based pufs,' *arXiv preprint arXiv:2308.06730*, 2023.

[196] B. Talukder, F. Ferdaus, and M. T. Rahman, 'A non-invasive technique to detect authentic/counterfeit sram chips,' *arXiv preprint arXiv:2107.09199*, 2021.

[197] M. Cortez, S. Hamdioui, and R. Ishihara, 'Design dependent sram puf robustness analysis,' in *2015 16th Latin-American Test Symposium (LATS)*, 2015, pp. 1–6. DOI: 10.1109/LATW.2015.7102498.

[198] N. Mishra, K. Pratihar, A. Chakraborty, and D. Mukhopadhyay, 'Modelling delay-based physically unclonable functions through particle swarm optimization,' *Cryptology ePrint Archive*, 2023.

[199] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber, 'Modeling attacks on physical unclonable functions,' in *Proceedings of the 17th ACM conference on Computer and communications security*, 2010, pp. 237–249.

[200] J. Delvaux, 'Machine-learning attacks on polypufs, ob-pufs, rpufs, lhs-pufs, and puf–fsms,' *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 8, pp. 2043–2058, 2019. DOI: 10.1109/TIFS.2019.2891223.

[201] M. Uddin, M. B. Majumder, and G. S. Rose, 'Robustness analysis of a memristive crossbar puf against modeling attacks,' *IEEE Transactions on Nanotechnology*, vol. 16, no. 3, pp. 396–405, 2017. DOI: 10.1109/TNANO.2017.2677882.

[202] N. Wisiol, G. T. Becker, M. Margraf, T. A. Soroceanu, J. Tobisch, and B. Zengin, 'Breaking the lightweight secure puf: Understanding the relation of input transformations and machine learning resistance,' in *Smart Card Research and Advanced Applications: 18th International Conference, CARDIS 2019, Prague, Czech Republic, November 11–13, 2019, Revised Selected Papers 18*, Springer, 2020, pp. 40–54.

[203] J.-L. Danger, S. Guilley, M. Pehl, S. Senni, and Y. Souissi, 'Highly reliable pufs for embedded systems, protected against tampering,' in *International Conference on Industrial Networks and Intelligent Systems*, Springer, 2021, pp. 167–184.

[204] T. Kroeger, W. Cheng, S. Guilley, J.-L. Danger, and N. Karimi, 'Enhancing the resiliency of multi-bit parallel arbiter-puf and its derivatives against power attacks,' in *Constructive Side-Channel Analysis and Secure Design: 12th International Workshop, COSADE 2021, Lugano, Switzerland, October 25–27, 2021, Proceedings 12*, Springer, 2021, pp. 303–321.

[205] T. Kroeger, W. Cheng, S. Guilley, J.-L. Danger, and N. Karimi, 'Making obfuscated pufs secure against power side-channel based modeling attacks,' in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2021, pp. 1000–1005.

[206] F. Ganji, *On the learnability of physically unclonable functions*. Springer, 2018.

[207] A. Vijayakumar, V. C. Patil, C. B. Prado, and S. Kundu, 'Machine learning resistant strong puf: Possible or a pipe dream?' In *2016 IEEE international symposium on hardware oriented security and trust (HOST)*, IEEE, 2016, pp. 19–24.

[208] F. Ganji, D. Forte, and J.-P. Seifert, 'Pufmeter a property testing tool for assessing the robustness of physically unclonable functions to machine learning attacks,' *IEEE Access*, vol. 7, pp. 122 513–122 521, 2019.

[209] P. Santikellur and R. S. Chakraborty, 'Correlation integral-based intrinsic dimension: A deep-learning-assisted empirical metric to estimate the robustness of physically unclonable functions to modeling attacks,' *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 10, pp. 3216–3227, 2021.

[210] M. Spain, B. Fuller, K. Ingols, and R. Cunningham, 'Robust keys from physical unclonable functions,' in *2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, IEEE, 2014, pp. 88–92.

[211] M. I. Khan, S. Ali, A. Al-Tamimi, A. Hassan, A. A. Ikram, and A. Bermak, 'A robust architecture of physical unclonable function based on memristor crossbar array,' *Microelectronics Journal*, vol. 116, p. 105 238, 2021.

[212] B. T. Bosworth *et al.*, 'Unclonable photonic keys hardened against machine learning attacks,' *APL Photonics*, vol. 5, no. 1, p. 010 803, 2020.

[213] A. Schaub, O. Rioul, and J. J. Boutros, 'Entropy estimation of physically unclonable functions via chow parameters,' in *2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, IEEE, 2019, pp. 698–704.

[214] N. Wisiol and N. Pirnay, 'Short paper: Xor arbiter pufs have systematic response bias,' in *Financial Cryptography and Data Security: 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10–14, 2020 Revised Selected Papers 24*, Springer, 2020, pp. 50–57.

[215] N. Wisiol, B. Thapaliya, K. T. Mursi, J.-P. Seifert, and Y. Zhuang, 'Neural-network-based modeling attacks on xor arbiter pufs revisited,' *Cryptology ePrint Archive*, 2021.

[216] P. H. Nguyen, D. P. Sahoo, C. Jin, K. Mahmood, U. Rührmair, and M. van Dijk, *The interpose puf: Secure puf design against state-of-the-art machine learning attacks*, Cryptology ePrint Archive, Report 2018/350, https://ia.cr/2018/350, 2018.

[217] X. Yang, S. Khandelwal, A. Jiang, and A. Jabir, 'A modelling attack resistant low overhead memristive physical unclonable function,' in *2020 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2020, pp. 1–4. DOI: 10.1109/DFT50435.2020.9250762.

[218] J. Yang *et al.*, 'A machine-learning-resistant 3d puf with 8-layer stacking vertical rram and 0.014% bit error rate using in-cell stabilization scheme for iot security applications,' in *2020 IEEE International Electron Devices Meeting (IEDM)*, IEEE, 2020, pp. 28–6.

[219] M. I. Khan, S. Ali, A. A. Ikram, and A. Bermak, 'Optimization of memristive crossbar array for physical unclonable function applications,' *IEEE Access*, 2021.

[220] N. Wisiol *et al.*, 'Splitting the interpose puf: A novel modeling attack strategy,' *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 97–120, 2020.

[221] J. Tobisch, A. Aghaie, and G. T. Becker, 'Combining optimization objectives: New modeling attacks on strong pufs,' *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 357–389, 2021.

[222] N. Wisiol and M. Margraf, 'Why attackers lose: Design and security analysis of arbitrarily large xor arbiter pufs,' *Journal of Cryptographic Engineering*, vol. 9, pp. 221–230, 2019.

[223] T. Kroeger, W. Cheng, S. Guilley, J.-L. Danger, and N. Karimi, 'Effect of aging on puf modeling attacks based on power side-channel observations,' in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2020, pp. 454–459.

[224] K. Stangherlin, Z. Wu, H. Patel, and M. Sachdev, 'Enhancing strong puf security with nonmonotonic response quantization,' *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 31, no. 1, pp. 55–64, 2022.

[225] K. Kursawe, A. R. Sadeghi, D. Schellekens, P. Tuyls, and B. Škorić, *Reconfigurable Physical Unclonable Functions - Enabling Technology for Tamper-Resistant Storage*. Los Alamitos, CA, USA: IEEE Computer Society, 2009.

[226] H. Cook, J. Thompson, Z. Tripp, B. Hutchings, and J. Goeders, 'Cloning the unclonable: Physically cloning an fpga ring-oscillator puf,' in *2022 International Conference on Field-Programmable Technology (ICFPT)*, 2022, pp. 1–10. DOI: 10.1109/ICFPT56656.2022.9974597.

[227] S. Duan and G. Sai, 'Bti aging-based physical cloning attack on sram puf and the countermeasure,' *Analog Integrated Circuits and Signal Processing*, pp. 1–11, 2023.

[228] Y. Wang, C. Wang, C. Gu, Y. Cui, M. O'Neill, and W. Liu, 'Theoretical analysis of delay-based pufs and design strategies for improvement,' in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, IEEE, 2019, pp. 1–5.

[229] O. Rioul, P. Solé, S. Guilley, and J.-L. Danger, 'On the entropy of physically unclonable functions,' in *2016 IEEE International Symposium on Information Theory (ISIT)*, IEEE, 2016, pp. 2928–2932.

[230] N. Wisiol *et al.*, *pypuf: Cryptanalysis of Physically Unclonable Functions*, version v2, 2021. DOI: 10.5281/zenodo.3901410.

[231] Y. Xu, Y. Lao, W. Liu, Z. Zhang, X. You, and C. Zhang, 'Mathematical modeling analysis of strong physical unclonable functions,' *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 12, pp. 4426–4438, 2020.

[232] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai, 'Threshold voltage distribution in mlc nand flash memory: Characterization, analysis, and modeling,' in *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2013, pp. 1285–1290.

[233] B. Efron, 'Bayesian inference and the parametric bootstrap,' *The annals of applied statistics*, vol. 6, no. 4, p. 1971, 2012.

[234] B. Sudret, S. Marelli, and J. Wiart, 'Surrogate models for uncertainty quantification: An overview,' in *2017 11th European conference on antennas and propagation (EUCAP)*, IEEE, 2017, pp. 793–797.

[235] R. Schöbi, 'Surrogate models for uncertainty quantification in the context of imprecise probability modelling,' *IBK Bericht*, vol. 505, 2019.

[236] A. I. Khuri and S. Mukhopadhyay, 'Response surface methodology,' *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 2, no. 2, pp. 128–149, 2010.

[237] R. Van Den Berg *et al.*, 'Entropy analysis of physical unclonable functions,' *named-content content-type= ref-degree> Ph. D. dissertation, MSc. thesis</named-content>,< institution content-type= department> Dept. Math. Comput. Sci.</institution>,< institution content-type= institution> Eindhoven Univ. Technol.</institution>, Eindhoven*, 2012.

[238] C.-F. J. Wu, 'Jackknife, bootstrap and other resampling methods in regression analysis,' *the Annals of Statistics*, vol. 14, no. 4, pp. 1261–1295, 1986.

[239] W. Leong, M. Z. Abdullah, and C. Khor, 'Optimization of flexible printed circuit board electronics in the flow environment using response surface methodology,' *Microelectronics Reliability*, vol. 53, no. 12, pp. 1996–2004, 2013.

[240] S. Haag and R. Anderl, 'Digital twin–proof of concept,' *Manufacturing letters*, vol. 15, pp. 64–66, 2018.

[241] A. Agapie, A. Andreica, and M. Giuclea, 'Probabilistic cellular automata,' *Journal of Computational Biology*, vol. 21, no. 9, pp. 699–708, 2014.

[242] T. Toffoli and N. Margolus, *Cellular automata machines: a new environment for modeling*. MIT press, 1987.

[243] F. M. Reza, *An introduction to information theory*. Courier Corporation, 1994.