

The Art of SRAM Security: Tactics for Remanence-based Attack and Strategies for Defense

Jubayer Mahmud

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Computer Engineering

Matthew D. Hicks, Chair

Leyla Nazhandali

Michael S. Hsiao

Mengjia Yan

Wenjie Xiong

April 04, 2024

Blacksburg, Virginia

Keywords: On-chip SRAM attacks, aging side channel, data remanence

Copyright 2024, Jubayer Mahmud

The Art of SRAM Security: Tactics for Remanence-based Attack and Strategies for Defense

Jubayer Mahmud

(ABSTRACT)

The importance of securing hardware, particularly in the context of the Internet of Things (IoT), cannot be overstated in light of the increasing prevalence of low-level attacks. As the IoT industry continues to expand, security has become a more holistic concern, as evidenced by the wide range of attacks that we observed, from large-scale distributed denial-of-service attacks to data theft through monitoring a device's low-level behavior, such as power consumption. Traditional software-based security measures fall short in defending against the full spectrum of attacks, particularly those involving *physical tampering* with system hardware. This underscores the critical importance of proactively integrating attack vectors that encompass both hardware *and* software domains, with a particular emphasis on considering both the analog and digital characteristics of hardware. This thesis investigates system security from a hardware perspective, specifically examining how low-level circuit behavior and architectural design choices impact SRAM's data remanence and its implications for security. This dissertation not only identifies new vulnerabilities due to SRAM data remanence but also paves the way for novel security solutions in the ongoing "security arms race".

I present an attack, *volt boot*, that executes cold-boot style short-term data remanence in on-chip SRAM without using temperature effect. This attack exploits the fact that SRAM's power bus is externally accessible and allows data retention using a simple voltage probe. Next, I present a steganography method that hides information in the SRAM exploiting long-term data remanence. This approach leverages aging-induced degradation to imprint data in SRAM's analog domain, ultimately resulting in hidden and plausibly deniable in-

formation storage in the hardware. Finally, I show how an adversary weaponizes SRAM data remanence to develop an attack on a hardware-backed security isolation mechanism. The following provides a brief overview of the three major contributions of this thesis:

❶ ***Volt boot*** is an attack that demonstrates the vulnerability of on-chip SRAM due to the physical separation common in modern SoCs' power distribution networks. By probing external power pins (to the cache) of an SoC while simultaneously shutting down the main system power, *Volt boot* creates data retention across power cycles. On-chip SRAM can be a safe memory when the threat model considers traditional off-chip cold-boot-style attacks. This research demonstrates an alternative method for preserving information in on-chip SRAM through power cycles, expanding our understanding of data retention capabilities. *Volt boot* leverages asymmetrical power states (e.g., on vs. off) to force SRAM state retention across power cycles, eliminating the need for traditional cold boot attack enablers, such as low-temperature or intrinsic data retention time.

❷ ***Invisible Bits*** is a hardware steganography technique that hides secret messages in the analog domain of SRAM embedded within a computing device. Exploiting accelerated transistor aging, *Invisible Bits* stores hidden data along with system data in an on-chip cache and provides a plausible deniability guarantee from statistical analysis. Aging changes the transistor's behavior which I exploit to store data permanently (*i.e.*, long-term data remanence) in an SRAM. *Invisible Bits* presents unique opportunities for safeguarding electronic devices when subjected to inspections by authorities.

❸ ***UntrustZone*** utilizes long-term data remanence to exfiltrate secrets from on-chip SRAM. An attacker application must be able to read retained states in the SRAM upon power cycles, but this needs changing the security privilege. Hardware security schemes, such as ARM TrustZone, erase a memory block before changing its security attributes and releasing it to other applications, making short-term data remanence attacks ineffective. That is, attacks such as *Volt boot* fail when hardware-backed isolation such as TEE is enforced.

UntrustZone unveils a new threat to all forms of on-chip SRAM even when backed by hardware isolation: long-term data remanence. I show how an attacker systematically accelerates data imprinting on SRAM's analog domain to effectively burn in on-chip secrets and bypass TrustZone isolation.

The Art of SRAM Security: Tactics for Remanence-based Attack and Strategies for Defense

Jubayer Mahmud

(GENERAL AUDIENCE ABSTRACT)

In computing systems, hardware serves as the fundamental bulwark against security breaches. The evolution in software security has compelled adversaries to seek potential vulnerabilities in the hardware. The infamous *cold boot* attack exemplifies such vulnerabilities, showcasing how adversaries exploit hardware to access runtime secrets, even when cryptographic algorithms protect the system's disk. In this attack, volatile main memory (DRAM) is 'frozen' at extremely cold temperatures, allowing it to retain information even when disconnected from the victim machine. Subsequently, an adversary transfers this 'frozen memory' to another machine to extract the victim's secrets. This classic case is among numerous sophisticated hardware vulnerabilities identified in recent years, highlighting the evolving challenge of securing hardware against ingenious attacks. This rise in hardware-based attacks across industry and academia underscores the importance of adopting a comprehensive approach to safeguard computing systems. This approach must encompass secure processor design, ensuring a trusted distribution chain, rigorous software security vetting, and protection against runtime side-channel leakage. Consequently, there is a growing emphasis in both industry and academia on prioritizing security in design decisions. My dissertation delves into the low-level hardware behaviors, particularly focusing on the data remanence phenomena of Static Random Access Memory (SRAM). By discovering new security vulnerabilities and proposing effective mitigation strategies, this thesis contributes to the ongoing effort to fortify computing systems against evolving threats that are rooted in the hardware.

SRAM stands as a ubiquitous form of volatile memory found in most processors and micro-controllers, serving as a crucial component for temporary storage of instructions and data to facilitate rapid access. By design, SRAM forgets its contents upon a processor’s power cycle and defaults to a state determined by low-level circuit behavior. However, this dissertation unveils the possibility of retaining on-chip information even after power cycling, leveraging inherent low-level circuit behaviors to create data retention. This revelation exposes major security implications, resulting in the following three key contributions:

Firstly, I introduce the *volt boot* attack, which exploits the vulnerability of on-chip SRAM, particularly to physical separation in modern System on Chip (SoC) power distribution networks. We conventionally assume on-chip SRAM secure against off-chip cold-boot attacks, but *volt boot* demonstrates the feasibility of achieving a similar state without traditional prerequisites such as low temperatures or long intrinsic data retention times.

Subsequently, I propose a data hiding technique—*invisible Bits*, which leverages accelerated device wear out to embed data into the transistors of SRAM. This method introduces a novel form of hardware-based steganography, concealing data within the analog domain alongside digital system data.

Lastly, I show how accelerated device aging can be weaponized to design a sophisticated attack aimed at extracting secrets from a Trusted Execution Environment (TEE) like ARM TrustZone. While short-term data remanence attacks such as *Volt boot* are rendered ineffective against hardware-backed isolation enforced by TEEs, *UntrustZone* harnesses the methodologies and tools from preceding works to induce long-term data remanence. This poses a new threat to on-chip cryptography that stores secrets *on chip*, even when fortified by hardware isolation mechanisms, such as ARM TrustZone.

Dedication

To Bangladeshi farmers, rickshaw pullers, and garment workers, whose hard work and sacrifices paid for my college (BUET)

Acknowledgments

I reached out to Matthew Hicks, my now advisor, via email after learning that my previous professor—who came to Virginia Tech for— was relocating. As an international student, this was a challenging time for me. Following our initial meeting, I asked Matt to be my advisor. He graciously accepted and has since played a crucial role in shaping my academic journey. In hindsight, this was one of the best decisions of my life. I am deeply grateful for his mentorship and support, without which this dissertation would not have been possible. Beyond academia, Matt has shown genuine care for my overall growth and development. His constructive criticisms in the gym and relentless optimism, especially in the face of harsh feedback from reviewers, have been invaluable. Next, I extend my gratitude to the members of my doctoral committee, Dr. Michael S. Hsiao, Dr. Leyla Nazhandali, Dr. Mengjia Yan, and Dr. Wenjie Xiong, for their generosity with their time, invaluable guidance, and insightful feedback.

I am immensely grateful to my parents for their relentless love and support. Despite having only completed middle school themselves, they have always pushed me to strive for excellence and reach beyond my perceived limits. Thank you for being my constant source of encouragement and for giving me the freedom to pursue my passions without reservation.

My friends have served as my most influential mentors. Their support has been a constant source of strength, no matter where life has taken me. I am incredibly fortunate to have such supportive friends by my side. I extend my heartfelt gratitude to Faruk, Sujon, Showrov, Anik, Ayon, Kanon, Shakill, Shuvo, Shuvro, Ashab, Dr. Sudipta Paul, Dr. Adnand

Rakin, and Dr. Naimul Hassan for their friendship and encouragement. I am who I am because of you all.

I extend my heartfelt gratitude to the wonderful friends I made during my time in graduate school. I am immensely thankful to Dr. Michael Moukarzel, Dr. Stefan Nagy, Harrison, Gagan, Mohammad, Daniel, Rishi, Prakhar, and all my current and former labmates for the invaluable lessons and camaraderie shared within the lab. Beyond the academic sphere, I owe a debt of gratitude to Maruf, Sefat, Murad, Imitaj, Mahir, Rafi, and many others who have become like family to me, turning Blacksburg into my home over the past five years.

Despite my home being 8500 miles away, I am incredibly fortunate to have found a sweet and loving family here. I express my deepest appreciation to my girlfriend Kiara and our totally awesome Doberman Pinscher, Bella, for their companionship throughout my Ph.D. You were there for me whenever I needed you. You have transformed this small town in the Blue Ridge Mountains into a place that feels like home, and for that, I am deeply thankful.

Funding acknowledgments: The project depicted is sponsored by the Defense Advanced Research Projects Agency (DARPA). The content of the information does not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred. Approved for public release; distribution is unlimited.

Contents

List of Figures	xv
List of Tables	xx
1 Introduction	1
2 Literature Review	6
3 Attacking on-chip SRAM Exploiting Power Domain Separation	10
3.1 Introduction	10
3.2 Background	13
3.2.1 Fully On-Chip Computation	13
3.2.2 Power Domain Separation	14
3.3 Cold Booting On-chip SRAM is Ineffective	15
3.4 Attack Model	18
3.5 Executing <i>Volt boot</i> Attack	19
3.5.1 Inducing Data Retention	19
3.5.2 Attack Enablers	20
3.6 Evaluating <i>Volt boot</i>	22

3.6.1	Attack Execution Steps	23
3.6.2	How Much Memory is Accessible to an Attacker?	26
3.7	Attack Execution in Different Memories	27
3.7.1	Attacking Caches	28
3.7.2	Attacking CPU Registers	31
3.7.3	Attacking iRAMs	32
3.8	Countermeasures	33
3.9	Related Work	35
3.10	Conclusion	36
4	Hiding Secret Messages in SRAM's Analog Domain	38
4.1	Introduction	38
4.2	Background	42
4.2.1	Influencing Power-on Behavior	42
4.3	Threat Model	44
4.4	Design	46
4.4.1	Message Pre-Processing	47
4.4.2	SRAM Analog-Domain Payload Encoding	48
4.4.3	Decoding Payloads via SRAM Power-on State	48
4.4.4	Decoded Payload Post-Processing	49

4.5	Evaluation	50
4.5.1	Error Profiling	52
4.5.2	Reducing Error	56
4.5.3	Performance Comparison	58
4.6	Invisible Bits in Action	60
4.7	Discussion	63
4.7.1	Multiple-Snapshot Adversary	63
4.7.2	Aging Complex Systems	64
4.7.3	Device Selection & Capacity Considerations	65
4.7.4	Adversarial Aging to Inject Noise	66
4.8	Related work	67
4.8.1	Hardware-based steganography	67
4.8.2	System-Level Steganography	68
4.8.3	Infrastructure-Level Steganography	69
4.9	Conclusion	69
5	Exploiting Long-term Data Remanence to Exfiltrate Secrets from ARM Trust- Zone Guarded SRAM	70
5.1	Introduction	70
5.2	Background	74
5.2.1	ARM TrustZone	75

5.2.2	SRAM Power-on State	76
5.2.3	Analog-domain Changes and Power-on State	77
5.3	Attack Overview	79
5.4	Identifying Attack Enablers	80
5.5	Attack Evaluation	84
5.5.1	Setup and Target Devices	84
5.5.2	Tuning Stress Conditions to Enhance Accuracy	88
5.6	Attack #1: Exfiltrate an AES key from TrustZone	90
5.7	Attack #2: Exfiltrate Proprietary Firmware from TrustZone	93
5.8	Attack #3: Exfiltrate Secrets from Cache	98
5.8.1	Accessing Cache's Power-on State	99
5.8.2	Attacking ARM Core's L1 Cache	100
5.9	Unifying the Attack Scenarios	100
5.10	Countermeasures	102
5.10.1	Software Defenses	102
5.10.2	Hardware Defenses	103
5.11	Discussion	105
5.12	Related work	106
5.12.1	Exploiting Voltage and Frequency Management Hardware	106
5.12.2	Exploiting Cache and DRAM	106

5.12.3 Exploiting Configurable Hardware	107
5.13 Conclusion	108
6 Conclusion and Future Work	109
6.1 Future work	110
6.1.1 Malicious Software-induced Aging Faults	110
6.1.2 SRAM Data Remanence-based Physical Attack Mitigation	112
Bibliography	114

List of Figures

2.1	Schematic diagram of an SRAM cell.	6
2.2	Internal structure of a 6T SRAM cell.	8
3.1	A simplified block diagram of an SoC's power domains. The PMIC is an external component that maintains a specific voltage level at each power domain supply pin.	15
3.2	Data cache (L1) snippet ($WAY0 = 256 \times 512 = 16KB$) of a Cortex-A72 core when the power is disconnected for a few milliseconds at $-40^{\circ}C$. An equal number of 1s and 0s in the cold-booted cache image indicates that the cache reset to its power-on state, <i>i.e.</i> , no data remained in the cache.	18
3.3	Typical power supply system. The LC combination supply lines are for switching regulators (right side), whereas the decoupling capacitor (left side) filters out noise during fluctuating load driven by LDOs.	20
3.4	Attack flow.	21
3.5	Pictures of the evaluation platforms showing the test points where voltage probes are attached. The platforms include (a) Raspberry Pi 4, (b) Raspberry Pi 3, and (c) i.MX535.	24

3.6	Snapshots of i-cache after attacking bare-metal software in (a) BCM2711 and (b) BCM2837 SoCs. Uninitialized cache cells power on into random state (see Figure 3.2), but when I execute <i>Volt boot</i> attack, instructions stay in the i-cache across power cycles.	27
3.7	Snapshots of the caches after executing <i>Volt boot</i> on a system running a general application. I generate the cache images from one WAY of each type of cache.	29
3.8	Visual representation of iRAM's data extraction for address (a) <code>0xF8000000</code> to <code>0xF8007FFF</code> (b) <code>0xF8008000</code> to <code>0xF800FFFF</code> (c) <code>0xF8010000</code> to <code>0xF8017FFF</code> (d) <code>0xF8018000</code> to <code>0xF8020000</code>	32
3.9	Hamming distance between image binary and post-attack binary.	35
4.1	<i>Invisible Bits</i> and its various use cases on SRAM embedded in Texas Instruments MSP432 [71]	39
4.2	(a) A schematic diagram of an SRAM cell (b) simulation waveform of startup state of the cell.	43
4.3	Illustrates (a) startup distribution of an SRAM, data directed aging (b) when an SRAM holds logic 0 , (c) logic 1 during stress. (d) Illustrates accelerated aging using supply voltage and operating temperature.	45
4.4	<i>Invisible Bits</i> overview.	46
4.5	(a) Schematic diagram of the evaluation setup. (b) Evaluation platform. The red board is an MSP432P401 launchpad and the green board is my custom control board.	51

4.6	Influence of stress time on error.	53
4.7	Normalized errors in <i>Invisible Bits</i> when stress-induced degradation recovers naturally.	55
4.8	Visual example of how a repetition code removes error from the decoded message.	56
4.9	Error reduction due to changing stress time and the number of copies used in the repetition code.	57
4.10	Application of Hamming(7,4) and repetition codes show how I can improve the error performance of a message hiding scheme that uses <i>Invisible Bits</i>	58
4.11	(a) Hiding information as plain-text, and ciphertext. (b) Shannon's entropy of SRAMs' power-on state. The normalized (by the number of symbols) entropy of an SRAM's power-on state is 0.0312, and the entropy becomes 0.0195 and 0.0312 in SRAMs that hide messages as plain-text and encrypted text, respectively.	61
4.12	An end-to-end steganography system using <i>Invisible Bits</i>	62
4.13	Distribution of Hamming weights captured at different phases and times of a covert communication.	64
4.14	Error and capacity trade-off. I encode a single copy message in each device and simulate Bernoulli trials for different payload copies and Hamming(7,4) error correction code.	66

5.1	An abstract view of a TrustZone-capable processor and its different access permissions throughout the system. In an ARMv8-A architecture, caches are shared between <i>Secure World</i> and <i>Normal World</i> . ARMv8-M architecture allows configurable partitions in the processor’s memory map.	74
5.2	(a) Schematic of a conventional 6-transistor SRAM cell and (b) power-on behavior of an SRAM cell at the 22nm technology node.	76
5.3	Heatmap of a SAML11 device’s power-on SRAM state.	77
5.4	Illustrates the target system’s (a) nominal power supply configuration where a switching regulator supplies the SoC’s cores. (b) My attack configuration, where I remove the inductor (L) and connect an external voltage probe to control the victim’s voltage.	81
5.5	A diagram illustrating the high-level configuration of the experimental setup.	85
5.6	Pictures of my experimental setup. Detachable target hardware sits on a CW308 baseboard [68] (bottom right), which is connected to the power-control board and debug hardware (top right).	87
5.7	(a) Response of a SAML11 chip when exposed to stress at different stress conditions with extremely biased software (i.e., all 0’s). (b) Stress response of cells with different logic states.	89
5.8	An attack model for <i>Secure World</i> AES key extraction.	90
5.9	Heatmap of the pre-stress memory dump (left) and post-stress retrieved information (right). Note that the first 2KB of the SRAM is reserved for the on-chip ROM.	94

5.10 Data-directed aging response of an SRAM cell. When an SRAM cell stores the complement of its power-on state during stress, the post-stress power-on logic remains unchanged (dotted green lines). If it stores the same logic as its power-on state, the post-stress power-on logic flips (solid green lines). Contrary behavior (red lines) causes post-stress data interpretation errors. .	94
5.11 Post-stress fractional Hamming distance in each word (32-bit) of the SRAM (address base 0x20000000). <i>CPU1</i> 's firmware is within 0x20012800 to 0x2001545C, showing how firmware burns-in in the SRAM's power-on state.	96
5.12 SRAM banks in the LPC device show different levels of stress-induced degradation based on their locations inside the die.	98

List of Tables

3.1	Errors in d-cache data after a cold boot attack execution in a BCM2711 SoC. A mean error is computed for each core at different temperatures. The fractional Hamming distance between cache content after the power cycle and cache's startup state is ~ 0.10 , indicating no data retention.	17
3.2	Evaluated platforms and their SoCs.	20
3.3	<i>Volt boot</i> evaluation platforms, test pads, and their nominal voltages.	24
3.4	Extracted data from d-cache of a BCM2711 SoC using <i>Volt boot</i> attack. The size of the d-cache in this SoC is 32KB, which is divided into two ways, <i>W0</i> and <i>W1</i>	30
4.1	A list of tested devices.	52
4.2	Spatial autocorrelation of power-on states of two SRAMs before and after stress. We stress each SRAM with one logic value (0/1); therefore, the post-stress power-on state reflects spatially-correlated errors in encoding.	54
4.3	Comparison among on-chip information hiding techniques.	59
4.4	Summary of the results when I encode a message in different devices.	60

4.5	Spatial autocorrelation and mean power-on bias of SRAM embedded within an MSP432. Layering encryption on top of <i>Volt boot</i> makes devices with no hidden message indistinguishable from devices with messages encoded. Without encryption, devices with messages encoded have biases in their power-on state (e.g., more 1s than 0s) and spatially-correlated values. <i>p</i> -values are $\ll 0.05$ for all the measurements.	62
5.1	A list of devices that we verified are susceptible to our long-term data remanence attack.	86
5.2	Effect of stress time on the retrieved information accuracy in (a) an LPC55S69 [122] device. and (b) a SAML11 [114] device.	89
5.3	The memory map of my system. A secure application (SW1) uses the memory area marked as secure and NSC, whereas the non-secure area belongs to the non-secure application (SW2).	91
5.4	The cell power-on state changes observed in AES key retrieval attack.	91
5.5	Memory partitions between <i>CPU0</i> and <i>CPU1</i>	95
5.6	Pre-stress and post-stress cell transitions of three LPC devices.	96
5.7	Firmware extraction accuracy for three target devices.	96

Publications from this Thesis

- **Jubayer Mahmud** and Matthew Hicks, "*SRAM Has No Chill: Exploiting Power Domain Separation to Steal On-chip Secrets*", International Conference on Architectural Support for Programming Languages and Operating Systems (**ASPLOS**), 2022. [104]
- **Jubayer Mahmud** and Matthew Hicks, "*Invisible Bits: Hiding Secret Messages in SRAMs Analog Domain*", International Conference on Architectural Support for Programming Languages and Operating Systems (**ASPLOS**), 2022. [103]
- **Jubayer Mahmud** and Matthew Hicks, " *UnTrustZone: Systematic Accelerated Aging to Expose On-chip Secrets*. 2024 IEEE Symposium on Security and Privacy (**Oakland**), 2024 [105].

Responsible Disclosure: Following standard vulnerability disclosure guidelines, I reported our findings to ARM as well as the manufacturers of the devices against which I validated UntrustZone, including NXP Semiconductors and Microchip Technology. I made this work public only after ARM released an architecture security advisory.

Chapter 1

Introduction

As outlined by the World Economic Forum, hardware integrity is fundamental to establishing digital trust in technology [91]. Over the last decade, the landscape of hardware-oriented system security has dramatically transformed, driven by the advent of sophisticated physical and microarchitectural attacks [23, 89, 96, 164]. These events have highlighted various low vulnerabilities at the hardware level, propelling the development of defensive measures across the entire computing hierarchy, from transistors to system software. However, despite these innovations, security measures often remain inadequate, challenged by the increasing complexity of systems and the continuous advancement of cyber threats. This dynamic situation emphasizes a pivotal shift in the cybersecurity domain: as enhancements in software security become more effective, attackers pivot their focus towards the hardware layer, unveiling a realm fraught with both challenges and untapped opportunities for enhancing system security. Against this backdrop, my dissertation investigates into the low-level aspects of Static Random-Access Memory (SRAM), a ubiquitous component in computing systems, to uncover vulnerabilities and provide security solutions.

The critical role of transistor-level characteristics in system security is significant, as a mere binary perspective of semiconductor memory (focusing solely on digital 1s and 0s) inadequately addresses the security implications inherent to these components. An analog property known as data remanence, where memory cells retain information beyond their intended lifespan, underscores this issue. This phenomenon is exemplified by the *cold boot* attack, which exploits data remanence in DRAM. Under normal conditions, DRAM cells require refresh cycles every 64 milliseconds to prevent data loss. However, the cold boot attack revealed that exposing DRAM to extremely low

temperatures could significantly prolong data retention, allowing attackers to extract sensitive information by transferring the DRAM module to another machine. This attack, by leveraging the reduced parasitic discharge rate of DRAM cells, thereby extending data retention far beyond expectations, proves the tangible risks of overlooking analog properties in security analyses. Before the discovery of such vulnerabilities in volatile memory, the concept of swapping memory devices to access secrets was primarily associated with mass storage devices like disks. Data-remanence attacks like *cold boot* and other physical attacks on off-chip memory (*i.e.*, bus probing) led academia and industry to adopt on-the-fly encryption of off-chip memory-bound transactions regardless of the target memory type (volatile or non-volatile), which constricts the security perimeter within the processor chip.

The temporary data remanence described above ultimately results in secrets becoming irrecoverable as memory cells fully discharge over time, rendering attack execution unreliable and error-prone. However, semiconductor memories also exhibit a more persistent form of data remanence, stemming from device ‘aging’, a physical phenomenon that gradually alters underlying circuit behavior. This permanent data remanence allows the stored states to be inferred years after the data was digitally erased, introducing security threats in volatile and non-volatile memories. An example is Flash memories’ permanent data remanence in the analog domain even after erasure [176]. Flash memory, cells default to a ‘1’ state; altering this state to a ‘0’ requires modifying the charge states of the underlying transistors. Frequent reprogramming of Flash memory leaves indelible data traces, even after the information appears to have been removed in a digital sense. The long-term data remanence phenomenon is even more surprising in volatile memories, such as DRAM and SRAM, which by design lose data once power is disconnected. Yet, at the analog level, these memories can retain data years beyond power loss due to natural or adversarial device-level changes in their underlying circuit elements, highlighting the complex nature of data remanence and its implications for system security.

In this dissertation, I investigate potential means of effectively exploiting on-chip SRAM data remanence for security applications. To this end, I present the *Volt boot* attack, which targets on-chip

SRAM through leveraging the power domain separation of System-on-Chips (SoCs). Subsequently, I delve into exploring long-term data remanence and begin by developing a hardware steganography method, referred to as *Invisible Bits*. This approach leverages device wear out to imprint information in SRAM’s analog domain, ultimately resulting in hidden information that is plausibly deniable. Finally, I propose an attack that exploits aging-induced data remanence of on-chip SRAM to exfiltrate hardware-backed Trusted Execution Environment (TEE) known as ARM TrustZone.

The subsequent sections outline the research problems addressed by my thesis and detail its contributions.

Volt boot

Adversaries can inspect external memory and their buses to access a system’s secrets if the memory-bound transactions are not encrypted. One solution is to keep secrets on-chip in SRAM in plaintext format; by construction, on-chip SRAM is isolated from external probing and has little intrinsic capacitance, making it robust against cold boot and other external memory attacks.

While it is the case that SRAM protects against traditional cold boot attacks, in this work I show you that there is another way to retain information in on-chip SRAM across power cycles and software changes. This work designs an attack, *Volt boot*, that demonstrates a vulnerability of on-chip volatile memories due to the physical separation common to modern system-on-chip power distribution networks. *Volt boot* leverages asymmetrical power states (e.g., on vs. off) to force SRAM state retention across power cycles, eliminating the need for traditional cold boot attack enablers, such as low-temperature or intrinsic data retention time. Using several modern ARM Cortex-A devices, this work demonstrates the effectiveness of the attack in caches, registers, and iRAMs. Unlike other forms of SRAM data retention attacks, *Volt boot* retrieves data with **100%** accuracy—without any complex post-processing (Chapter 3).

Invisible Bits

Electronic devices are increasingly the subject of inspection by authorities. While encryption hides secret messages, it does not hide the transmission of those secret messages—in fact, it calls attention to them. Thus, an adversary, seeing encrypted data, turns to coercion to extract the credentials required to reveal the secret message. Steganographic techniques hide secret messages in plain sight, providing the user with plausible deniability, removing the threat of coercion.

This work unveils *Invisible Bits*, a new steganographic technique that hides secret messages in the analog domain of SRAM embedded within a computing device. Unlike other memory technologies, the power-on state of SRAM reveals the analog-domain properties of its individual cells. This work shows how to quickly and systematically change the analog-domain properties of SRAM cells to encode data in the analog domain and how to reveal those changes by capturing SRAM’s power-on state. Experiments with commercial devices show that *Invisible Bits* provides over 90% capacity—two orders-of-magnitude more than previous on-chip steganographic approaches while retaining device functionality—even when the device undergoes subsequent normal operation or is shelved for months. Experiments also show that adversaries cannot differentiate between devices with encoded messages and those without (Chapter 4).

UntrustZone

Modern systems allow hardware-backed security enclaves called Trusted Execution Environments (TEEs); TEEs add hardware-level protections on top of keeping secrets on chip that extend protection against privileged software and flaws within the untrusted parts of the software. While the best TEEs protect against concurrent and temporally recent attacks (e.g., the cold boot attack), this work aims to uncover a threat to all forms of on-chip crypto: SRAM’s long-term data remanence.

TEE, such as ARM TrustZone, prevents short-term data remanence attacks, such as the one I propose in Chapter 3, by erasing the contents of memory if there are security attribute changes. This countermeasure ultimately prevents short-term data remanence attacks. As the previously *Invisible Bits* shows, SRAM’s power-on state changes at the analog-domain level in a data-dependent way as

software uses it. The objective of this work is to propose a method for launching an attack capable of exfiltrating on-chip SRAM data that is safeguarded by ARM TrustZone. Additionally, the work aims to provide a set of countermeasures that can be employed to prevent such attacks (Chapter 5).

Chapter 2

Literature Review

This chapter introduces an overarching literature review and fundamental concepts essential for setting the stage for this thesis. Subsequent chapters (§3.9, §4.8, and §5.12) provide focused and detailed analyses of related work, positioning each project relative to its state-of-the-art.

Static Random Access Memory (SRAM) is, a ubiquitous memory technology, present in virtually every computing device, from embedded sensors to supercomputers. SRAM is known for its ultra-fast, low-power characteristics, which make it ideal as cache memory, registers, and main memory (in microcontrollers). Besides its intended application as memory, the hardware security research community developed numerous security primitives and protocols using SRAM. For example, SRAM-based physical unclonable functions (PUFs) can be used to create unique identifiers that can be used for secure authentication and encryption [63, 65, 142].

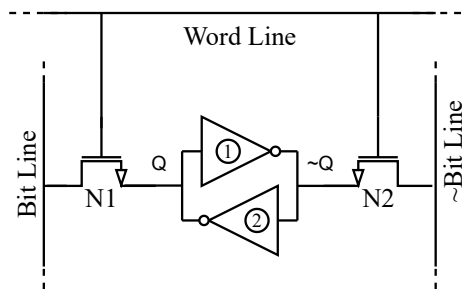


Figure 2.1: Schematic diagram of an SRAM cell.

SRAM-based PUFs exploit the random variations that occur in SRAM cells due to manufacturing imperfections, creating device-specific patterns that can generate secure keys and hardware fingerprints. In addition, SRAM can be used for memory encryption and secure bootstrapping. SRAM-

based memory encryption involves encrypting data in memory as it is written to the SRAM and decrypting it as it is read out, which can prevent unauthorized access to sensitive data from external memory during transactions and at reset [33, 190]. SRAM-based secure bootstrapping involves using the SRAM to store a trusted boot image, which can be used to ensure that the system boots up securely and has not been compromised.

My work explores on-chip SRAM security from the analog domain behavior of constituent transistors; more specifically, the security implication of short-term and long-term data remanence of SRAM cells. Before diving into these security implications, I briefly explain the fundamentals needed to understand the remanence properties of SRAM cells.

Figure 2.1 illustrates a typical 6-transistor SRAM cell, which is composed of two inverters in a positive feedback configuration to hold a data bit. The cell's state is accessible through transistors N1 and N2, and these transistors provide access to the data bit (Q) and its complement ($\sim Q$), respectively. Unless a processor executes a read/write command, *Word Line* remains de-asserted with data stored in the cross-coupled structure formed by inverter ① and inverter ②.

SRAM is energy-efficient, fast, long-life, and self-refreshing; the only requirement for data retention is sufficient voltage from the power supply to maintain the positive feedback loop between the two inverters. The voltage required by an SRAM cell to retain the state is called its data retention voltage [64]. Data retention voltage is both process variation and data-dependent but is generally much lower than the threshold voltage of either inverter (i.e., the 'turn-on' voltage). Provided the voltage of a cell is more than or equal to its data retention voltage, the cell retains its state. Exploiting this property, modern processors dynamically scale down the voltage when the RAM is not actively accessed because it reduces the *energy leakage* through parasitic paths.

Going down in the abstraction level, Figure 2.2 shows the conventional 6-transistor SRAM cell where I unrolled the inverters in their constituent transistors (transistors M1 and M2 form *Inverter-1*, while M3 and M4 form *Inverter-2*) and two access transistors (M5 and M6) that allow access to the logic state stored in the cell. At design time, aspect ratios of the *Inverter-1*'s PMOS and NMOS

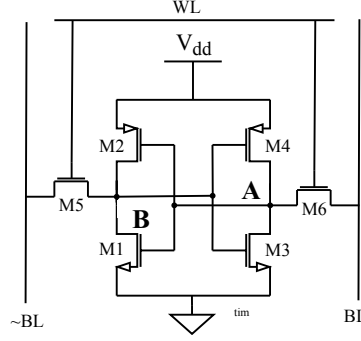


Figure 2.2: Internal structure of a 6T SRAM cell.

match perfectly with the PMOS and NMOS of *Inverter-2* [$(W/L)_2 = (W/L)_4$ and $(W/L)_1 = (W/L)_3$]. The symmetric design maximizes the static noise margin and minimizes the data retention voltage [65, 66]. Process variation causes post-manufacturing transistor mismatch—making $(W/L)_2 \neq (W/L)_4$ and $(W/L)_1 \neq (W/L)_3$; $v_{th2} \neq v_{th4}$ and $v_{th1} \neq v_{th3}$.

As a result of inverter asymmetry, one inverter turns on faster than the other, causing an SRAM cell to prefer one power-on state over another. When the cell is unpowered, all wires are at the ground voltage. As power is applied to the cell, both node A and B attempt to follow the supply voltage's (V_{dd}) ramp-up. After this initial mutual ramp-up, one of the transistors, M2 or M4, turns on faster due to differences in analog-domain properties, causing its output node (B or A, respectively) to pull up to the supply voltage, while the other node is pulled down to the ground by M3 or M1, respectively. Thus, the post-power-on steady-state voltage at nodes A and B depends on a hardware-level race condition between the inverters. The relative analog-domain properties (primarily threshold voltage) of the transistors largely determine the race winner.

SRAMs' ubiquity drives a large body of research that studies the implication of data retention properties across power cycles. One set of studies investigates how cooling affects the SRAM's data retention and shows how this is a potential exploit in cloning physical unclonable functions and leaking secrets [6, 7, 22, 58, 160]. These studies report that SRAM's data retention can be as high as $\approx 80\%$ when removed from power for 20ms at -110°C , but post-reboot state retention becomes 0% when the temperature is lowered to -40°C [6]. Another set of studies exploits SRAM's

data imprinting effect for applications where a cell holds the same logic state for a long period of time [22, 98, 107, 182]. The fundamental idea behind these attacks is a natural phenomenon known as circuit aging. As software uses an SRAM cell, the circuit goes through analog-domain changes, revealing the logic state that was stored in the cell through its power-on state. These attacks involve a significant investment of time *and* other technical efforts to achieve reasonable data retrieval accuracy and require data to remain in the same SRAM cells with the same value for over a decade to have even modest data recovery. Note that, unlike DRAMs, SRAMs are bistable; therefore, an attacker needs to consider both the logic states for correcting any error in a piece of retrieved secret information.

Data retention across power cycles is commonly viewed as a security risk, however, extensive research has demonstrated that data remanence can be utilized in numerous useful applications within both systems and security domains, *e.g.*, True Random Number Generation (TRNG) [165] and time-keeping functionality in batteryless devices [132]. While SRAM is a volatile memory, similar to DRAM, underlying cell architecture makes the cells retain the data for a short time in the event of power failure. Drawing a parallel from the DRAM cells, researchers showed that SRAM, in fact, shows similar behavior for short-term data retention, *i.e.*, momentary data retention is possible across power cycles in SRAM cells [6]. However, DRAM cold boot attack strategy is ineffective in on-chip SRAM as attack execution is difficult and error-prone due to its embedded nature and short retention time.

Chapter 3

Attacking on-chip SRAM Exploiting Power Domain Separation

3.1 Introduction

An increasingly connected world makes us dependent on computing devices that handle a wide range of security- and privacy-critical operations. On the personal side, we use smartphones and watches to manage bank transactions and store identity information. On the industrial and government side, embedded devices monitor remote system operations and feed data critical to the industrial processes and national defense. Physical access to these devices leads to a wide range of security exploits, including impersonation, proprietary software cloning, and infiltration and exploitation of industrial and defense infrastructures.

The most common approach to prevent data loss from a system's non-volatile memory is to encrypt it using full disk encryption methods, such as BitLocker [116] and VeraCrypt [88]. These encryption methods protect user data using a password or PIN so that even if a device is lost or stolen, the non-volatile memory remains inaccessible to an attacker. An encrypted disk demands user authentication whenever there is a reboot, providing adequate security even if an attacker removes it physically from a system and attempts to access its content from another machine.

Password-protected disk encryption methods force attackers to exploit other types of memories, such as DRAMs. Halderman *et al.* show how an attacker gains access to a disk encryption key

by *cold booting* a system and dumping its main memory [59]. In this attack, the authors use low temperature (-50°C) to ‘freeze’ the data in DRAM cells so that even if the memory is *out of power* for a short time, it retains its logic states. Once ‘frozen’, an attacker physically removes and inserts the victim DRAM in another machine to run forensics on the dumped memory image. From this point, it is trivial to post-process and extract security- and privacy-critical information from a system’s main memory. While this attack is practical for larger devices where DRAM is removable (e.g., laptops), it poses a few technical challenges for mobile and other embedded devices. In an embedded device, the memory chips and processors are soldered on the PCBs, making it difficult to remove them from a system. *FROST* [119] overcomes this challenge by allowing device reset to factory default—preserving DRAM’s content while the device boots from another media.

To defend devices from cold boot attacks, researchers proposed numerous methods where sensitive code and data remain encrypted in on-chip memory; decryption occurs only in the on-chip memories such as caches [120, 190]. Since SoCs already contain large enough on-chip storage to hold keys and cryptographic states, executing software on these memory requires no additional hardware. For example, *TRESOR* uses x86 debug registers to store sensitive AES states without leaking security-critical information to off-chip memory [120]. Researchers extend this idea to ARM devices where the CPU fetches the encrypted software and data to the on-chip memories before decryption and execution [30, 31, 49, 50, 157, 190]. These works advocate fully on-chip execution for cryptographic operations because attacking a processor’s internal memory is expensive and demands sophisticated attack methods, such as decapsulation. This work evaluates the security of on-chip computation schemes, specifically under the cold boot attack model. I empirically show that these methods are secure from the traditional cold boot attack [59], but the power domain separation common to modern SoCs allows us to create a similar effect, which exposes on-chip secrets to physical attackers.

On-chip memories are mostly SRAMs and built into the processor die itself (e.g., caches), which alone makes it much more secure than off-chip memories against physical attackers (e.g., probing attacks [62, 190]). Previous work shows that SRAM partially retains its state for a few milliseconds

in extremely low temperature ($< -110^{\circ}\text{C}$) [6], exposing a system to such low-temperature risks damaging its other components, such as the battery—not to mention the challenge of creating such a low temperature. In addition, attacking embedded devices requires a physical power disconnect which is typically much longer than SRAM’s data retention time.¹

I present *Volt boot*, a method that executes a physical memory disclosure attack on SRAM-based on-chip memories by exploiting power supply domain separation in SoCs. Different physical blocks in an SoC require a specific voltage level to meet performance demands while reducing power requirements. These blocks are divided into different domains that need external active and passive circuit components, such as a Power Management Integrated Circuit (PMIC), capacitors, and inductors. The circuit design of these domains allows fine-grained control of the system’s total energy budget and performance. From a power management perspective, these domains are independent and allow full power down at runtime when not needed by the system. I uncover an attack vector in such design choice that is exploitable for a cold-boot-style attack, eliminating the dependency on low temperature and an SRAM’s intrinsic data retention time.

In summary, this work makes the following technical contributions:

- I demonstrate that traditional cold-boot-style attacks, which leverage low-temperature-induced data retention, are ineffective on embedded SRAMs (§3.3).
- I uncover a new attack vector that exploits *power domain separation* of modern SoCs and show that such *architectural design choice can be weaponized* for breaking fully-on-chip cryptography (§3.5).
- Using *three* commercially available Cortex-A profile devices, I demonstrate the attack by retrieving data from caches (§3.7.1), CPU registers (§3.7.2), and iRAMs (§3.7.3). Provided that an application runs from internal memory (*e.g.*, cache) undisturbed by operating sys-

¹I define the intrinsic data retention time of an SRAM cell as the time it takes to discharge (and lose the data) when disconnected from power [132].

tem’s background process, I retrieve memory image with **100% accuracy**.²

- I survey potential countermeasures and their trade-offs (§3.8).

3.2 Background

As we have seen in chapter 2, embedded Static Random-Access Memory (SRAM) serves a variety of roles in modern computing systems including, storing microarchitectural states, buffering address translations, a cache for instructions and data, and as the main memory for many lightweight devices. Their embedded nature naturally protects from physical-level attacks that rely on interposing the interface between memory and the computation core (e.g., bus probing and cold boot attacks). This section discusses the fundamentals of on-chip SRAM, on-chip power networks, and the rationale behind on-chip cryptography.

3.2.1 Fully On-Chip Computation

Storing security-critical information in a DRAM (which is off-chip) is unsafe, motivating both industry and academic research to push for safer off-chip memory management schemes. Since Intel’s 6th generation processor, all subsequent processors obfuscate data in the DDR3 and DDR4 DRAMs using session keys and pseudo-random numbers [117, 186]. On the academic side, researchers propose fully SoC-bound computations where sensitive information leaves the chip only when encrypted. Modern SoCs contain a reasonably large volatile memory. In most cases, it is inessential to encrypt and decrypt every transaction; software can store intermediate states on on-chip memories in plain text. This idea inspired numerous on-chip computation methods. *Sentry* [31] and *Copker* [51] uses iRAMs and caches as temporary memory to avoid exposing secrets in DRAM. *Cache-assisted Secure Execution (CaSe)* extends this idea by adding Trust-Zone support to a partially

²The reported accuracy indicates data retention accuracy and not the information extraction accuracy. Exact information retrieval accuracy varies depending on the data present in an SRAM.

locked cache. The processor fetches encrypted software from main memory and stores it in a locked cache as plain text. From this point, the unencrypted software remains in the cache for the duration of execution. Similarly, *TRESOR* [120], *PRIME* [49], and *Security Through Amnesia* [157] use on-chip registers (*e.g.*, debug or multimedia) to implement cryptographic algorithms on-chip.

These methods essentially emulate a microcontroller’s behavior in a large-scale application processor and reduce the attack surface to the borders of the chip itself. Given no unencrypted data is released off-chip, these methods provide strong security against the most sophisticated physical memory disclosure attacks, *e.g.*, cold boot. While these methods induce performance penalties on other applications by partially blocking hardware resources, this work evaluates only their security aspects.

3.2.2 Power Domain Separation

SoCs contain a large number of circuit blocks with different analog characteristics. Because of performance and power efficiency constraints, these blocks operate under separate voltage domains. An SoC’s Power Management Unit (PMU) manages voltage required for specific domains at runtime, depending on each domain’s workload [4]. In a complex SoC, dozens of off-chip supply pins are connected to the power domains to regulate different analog circuit behavior, such as ground bounce, power-supply noise, and per-pin current supply. For this paper, I divide the SoCs power supply domains into three major areas (see Figure 3.1):

- **Core power domain:** the processing elements are in this domain. For example, the ARM cluster in a multi-core SoC draws power from the core supply voltage domain. This domain supplies power to the L1 caches and their associated control circuitry, besides computing elements such as CPU extensions and GPUs.
- **Memory power domain:** this domain supplies power to the memories and their associated peripherals. Most SoCs manage main memory, non-volatile memory (*e.g.*, Flash), and L2/L3

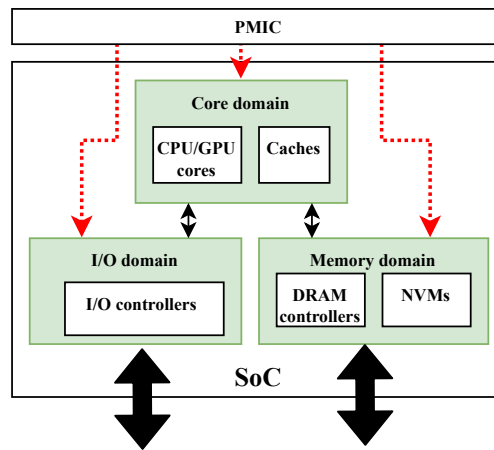


Figure 3.1: A simplified block diagram of an SoC's power domains. The PMIC is an external component that maintains a specific voltage level at each power domain supply pin.

caches with this power domain.

- **I/O power domain:** power for the I/O controllers and external peripherals is drawn from this domain.

SoC designers subdivide power domains into smaller logical blocks that allow for more fine-grain control of the different components within an SoC. For example, some processors allow powering down individual cache components in its L1 memory domain through software [92]. The domains are separated using power gating to balance energy consumption and performance of independent blocks at startup and runtime.

3.3 Cold Booting On-chip SRAM is Ineffective

DRAMs are typically separate IC packages externally connected through a memory bus, exposing these memory modules to different physical attacks, such as cold boot and bus probing. Given a system encrypts its code and data while 'at rest' in a disk and decrypts data while in volatile memories, capturing a live system and extracting the volatile memories leads to key disclosure, enabling secret exfiltration.

DRAM holds data as charge in a capacitor. Capacitors continually leak charge, which is why DRAM cells require periodic refresh operations to ensure data retention (typically every 64ms). In the absence of refresh, a DRAM cell's data retention time depends on the *time* it takes for capacitors to leak sufficient charge to change the digital value interpreted by sensing the voltage across the capacitor. Even without power, a just-refreshed DRAM cell will retain data for at least 64ms.

Temperature affects DRAM cell discharge rate; lower temperatures increase data retention time. Reducing the temperature of a memory device keeps the data in the memory for a short period even if the power is turned off, which is the fundamental idea behind the cold boot attack [59]. Both DRAM and SRAM partially retain their data across power cycles for a short time when the temperature is reduced below a certain level [6, 7, 59] due to their intrinsic capacitance (although DRAM retains for orders of magnitude more time due to its increased capacitance). Information retrieval accuracy of a *cold booted* SRAM cell depends on temperature, manufacturing technology, and power-off time. The following discusses the technical hurdles that prevent a cold boot attack on SRAM:

- **SRAM's placement:** As mentioned before, on-chip SRAM is tightly coupled with processing cores and are built into the processor die itself. Such placement of SRAM makes it inaccessible given the traditional physical attacker threat model. The *embedded nature* of SRAMs complicates the cooling process as well because an attacker needs to freeze the entire device. *Cold booting* this memory requires extreme low temperature and risks bricking of the system.
- **Short retention time:** A cold boot attack cannot be launched on embedded memories by resetting a device through software because a system easily prevents such attempts by purging residual memories as part of a core's power down sequence [26]. Disconnecting the power from a device is the only reliable way to prevent the system from executing any residual memory purging routines.

An abrupt power disconnect from a device while executing critical security operations ensures target information remains in SRAM. To power cycle an embedded device, one must

Temperature	0°C Recommended Min.	-5°C	-40°C SoC's hard limit
Error	50.14%	50.06%	50.39%

Table 3.1: Errors in d-cache data after a cold boot attack execution in a BCM2711 SoC. A mean error is computed for each core at different temperatures. The fractional Hamming distance between cache content after the power cycle and cache's startup state is ~ 0.10 , indicating no data retention.

manually disconnect its power supply (e.g., the battery). As the literature suggests, SRAM retains information for only a few milliseconds—even under extremely cold conditions, which is insufficient to execute a reboot by manually disconnecting power from a device [119]. In addition, SRAM is tightly coupled with the compute cores. An abrupt power disconnect draws energy from all parts of the SoC to the power-hungry processing elements, which accelerates the charge drain from SRAM's intrinsic capacitors. The combined effect of these SRAM-specific variables shortens the overall data retention time enough to make cold-boot-style attacks on SRAM infeasible.

- **Effect of low-temperature:** Typically, systems turn off when the operating temperature crosses a certain threshold set by the manufacturers. Executing a cold boot attack on SRAM requires extremely low temperature ($< -110^\circ$) [6], which is far beyond the operating limit of most devices. I reproduced a similar attack as *FROST* [119] in cache memory (SRAM, as opposed to the DRAM targeted in the original attack) of a Raspberry Pi 4 (a quad-core cortex-A72 device) [92] to study cold temperature data retention of its embedded SRAM. I load bare-metal software to populate both the d-cache and i-cache of each core and extract the cached data in a binary image. Then the device undergoes static cooling in a *TestEquity* thermal chamber [166] for an hour to stabilize the core temperature. I power cycle the device for a few milliseconds and extract the cache (§3.6) data to compare it to previously-stored binaries. Table 3.1 lists the *mean* mismatch between post-reboot retrieved cache and pre-stored binary for each core at different temperatures. The information retrieval errors indicate almost no data retention even at $-40^\circ C$. Note SRAMs boot up into random states where approximately 50% of the bits are 1s. Figure 3.2 illustrates a snippet of d-cache to

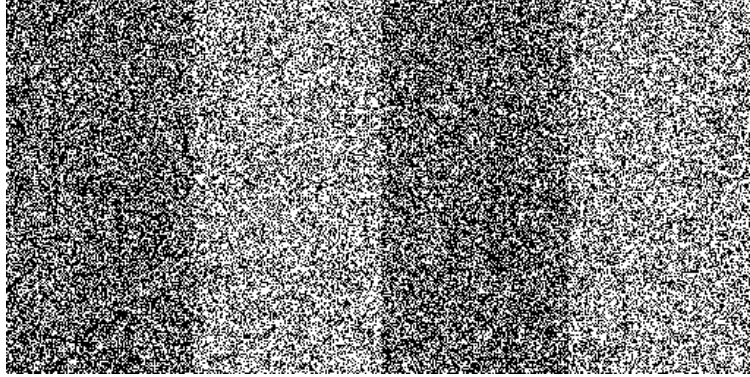


Figure 3.2: Data cache (L1) snippet ($WAY0 = 256 \times 512 = 16KB$) of a Cortex-A72 core when the power is disconnected for a few milliseconds at $-40^\circ C$. An equal number of 1s and 0s in the cold-booted cache image indicates that the cache reset to its power-on state, *i.e.*, no data remained in the cache.

provide a clear picture of the post-power-cycle state of an SRAM.

3.4 Attack Model

This work shows how power domain separation and exposed memory power supply pins induce artificial SRAM state retention, leading to a similar effect as a cold boot attack *without* requiring temperature control. Naturally, my attack model is based on the cold boot attack [59] where an attacker has physical access to a device. The most significant modifications to the original cold boot threat model are the location and type of the memory. My target is SRAM embedded within the core of a device, which prevents direct access to the memory contents without damaging the chip. This threat model is consistent with the threat model presented in the *FROST* [119], *Sentry* [31], and *CaSE* [190] systems. These papers consider a threat model where an attacker captures a device (*i.e.*, lost or stolen) that is protected against memory disclosure attacks using a lock screen and non-volatile memory encryption. Then the attacker boots the victim device from a media, *e.g.*, boot ROM or USB, to dump the uninitialized volatile memory image.

This threat model extends to headless embedded devices that collect, store, and transfer sensitive information in an unsupervised environment. Several system protection methods exist where de-

vices are permanently locked from programming or software updates. The behavior of such devices resembles an Application-Specific Integrated Circuit (ASIC), and I consider these systems out of the scope of this threat model.

3.5 Executing *Volt boot* Attack

Volt boot is an attack that exploits an SoC's power domain separation to induce cross-power-cycle data retention in embedded SRAM. In this section, I show circuit design choices made for power and performance reasons lead to a new cold-boot-style attack, but with increased precision and fewer requirements than traditional low-temperature-based data retention attacks.

3.5.1 Inducing Data Retention

SoCs need external pins to supply specific voltage to optimize performance and efficiently use energy under fluctuating loads. Usually, a PMIC supplies power to each domain in a particular order to bring up the board from reset. Figure 3.3 illustrates a typical power management IC used in modern SoCs. Generally, LDOs supply power to the domains where voltage fluctuation is limited, whereas domains with high load fluctuation (and dynamic voltage and frequency scaling) use switching regulators to save energy via heat loss (e.g., BUCK converters on the right side of the PMIC). When a power domain of an SoC draws a large current under the demand of software/hardware, the parasitic inductance of the board and the package drop the voltage at the supply lines; this is called droop. To counter droop, the supply voltage pins are extended out of the SoC to connect capacitors so that they 'absorb' the current surge, keeping supply voltage closer to nominal. Regardless of the type of regulator used in a power domain, the pins connected to the PMIC require passive components to filter out noise generated during load fluctuation.

Each power domain is power-gated to allow independent control at startup and run time. My

System on Chip	CPU core	Board	SRAM	Manufacturer
BCM2711	Quad-core Cortex-A72	Raspberry pi 4 Model B [47]	L1I: 48KB, L1D: 32KB, L2: 1MB	Broadcom
BCM2837	Quad-core Cortex-A53	Raspberry pi 3 Model B [46]	L1I: 32KB, L1D: 32KB, L2: 512KB	
i.MX535 [123]	Cortex-A8	i.MX53 Evaluation board	L1I: 32KB, L1D: 32KB, L2: 256KB, iRAM: 128KB	NXP Semiconductor

Table 3.2: Evaluated platforms and their SoCs.

observation is that if I externally maintain a steady voltage to the pin that supplies power to a target memory domain, it retains data—even while the rest of the system undergoes a power cycle. I maintain the domain voltage level from an external voltage probe while disconnecting the main supply line to the PMIC. That way *Volt boot* artificially creates SRAM data retention across a power cycle using a voltage probe in a PCB’s test pad or bare passive component’s lead that is connected to a target memory domain.

As discussed in Section 3.3, cold booting is an impractical way of attacking on-chip SRAMs. Even if it is possible to retain information across power cycles, the extracted data is erroneous because SRAM’s charge leakage follows a normal distribution, and so some cells will lose their data even for a short power disconnect [129]. Note that SRAM cells are bistable, which makes it harder to look for keys using the algorithm proposed in the original cold boot attack [59]. Since *Volt boot* attack involves no other physical variables (e.g., temperature), memory discharge rate or manufacturing technology node is irrelevant, which allows me to achieve 100% data recovery.

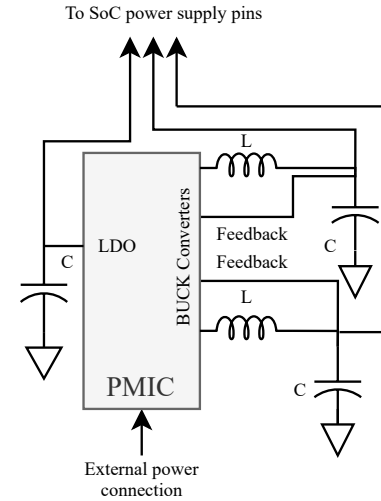


Figure 3.3: Typical power supply system. The LC combination supply lines are for switching regulators (right side), whereas the decoupling capacitor (left side) filters out noise during fluctuating load driven by LDOs.

3.5.2 Attack Enablers

While power domain separation serves as the basis for my attack, a combined effect of multiple aspects and at-

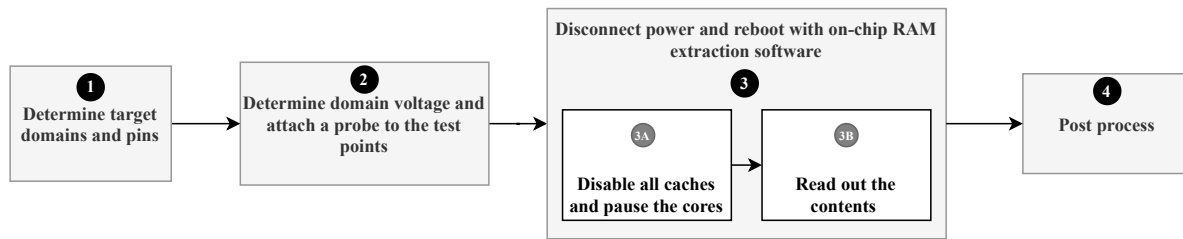


Figure 3.4: Attack flow.

tributes of a system enables *Volt boot*. I identify each attack enabler as follows:

Ubiquity of SRAM

SRAM is available in every computing device ranging from resource-constrained microcontrollers to server-class processors, which allows inducing artificial SRAM state retention in any SoC that has separate SRAM and compute core power domains.

Internal RAMs Stores Data in Plain-text

As mentioned in Section 3.2, cryptography application developers consider on-chip memories safer compared to external memories. As a result, unlike external DRAMs, scrambling or encrypting SRAM's data is uncommon in commodity processors. Thus, access to data residing in on-chip SRAM guarantees a plain-text version of the information, even if external memories implement scrambling or encryption.

Domain Specific Exposed Power Supply Pins

As discussed above, performance, energy efficiency, and die area reduction are the primary reasons to expose domain-specific voltage pins out of an SoC; some of these pins are connected to the embedded SRAM. By construction, data remains error-free in SRAM as long as its supply voltage remains above its data retention voltage (§3.2). Note, this voltage is well below the nominal

supply voltage of the power domain the cells are connoted to. The power domain separation through power gating methods and dedicated pins exposed from the SoCs let us shut down the entire system while the small portion that holds sensitive data ‘alive’.

No Default RAM Reset Hardware

SRAM stays at an uninitialized state after booting-up because of two main reasons: ① some SRAMs are large (>1MB) compared to the other on-chip resources, and resetting such large memory by iterating over the entire address (usually line by line for caches) space reduces boot speed significantly, and ② SRAM’s startup state has numerous security applications, such as PUF [140] and TRNG [63]. Note that cleaning and invalidating a cache at the boot phase does not erase the contents; these operations set the invalid bits to prevent a cache hit, but the data remains unchanged. The co-processor interface still allows reading out the cache contents from a proper exception level (EL3 for ARM devices). Therefore, initializing cache lines using a software interface is currently the only means to reset the power-on state of L1 caches, which needs the execution of DC ZVA instructions for every line [12, 92]. The purpose of this instruction is to allow initializing a large block of memory in the cache for a particular data structure without writing zeros in the external memories. Note cleaning/invalidation instructions apply to both instruction and data caches, but *resetting instructions* are exclusive to d-caches.

3.6 Evaluating *Volt boot*

An SoC’s domain-separated power management architecture allows us to supply voltage to a target memory through exposed pins and keep part of the chip active (i.e., retaining state) while the rest of the system resets. The actual method used to access embedded SRAM varies depending on the targeted SoC, but devices’ power supply methods are very similar at a high level. I evaluate *Volt boot* using ARM devices from different vendors, and to explore the generality of the attack,

I choose devices that span a broad range of applications. For example, Raspberry Pis represents a wide array of systems, ranging from headless embedded devices common in IoT applications to systems capable of running a full-fledged operating system. To expand the targeted memory types, I include an SoC designed for multimedia applications, because it contains iRAM. Table 3.2 lists the specification of the evaluation platforms. The SoCs in these systems draw power from three different power management devices, and I observe similar circuit design choices for the off-chip passive components (see Figure 3.3).

I provide a stable voltage to each SoC's target power domain using a bench power supply with >3A current driving capability. While attaching a probe at the same voltage level draws only a few milliamps, an abrupt power disconnect from the compute core supply line of the PMIC spikes the current momentarily on the target power domain. Such a current surge drops the voltage below the data retention voltage of SRAM (§3.2), leading to errors in the extracted information. Therefore, a power supply capable of supplying sufficient current is essential when the target memory domain also supplies power to the CPU core(s).

3.6.1 Attack Execution Steps

In this section, I discuss how to execute an attack on SoCs (see Figure 3.4 for a summary).

1. **Identifying target domains and their associated pins:** Once I identify a target device, the first step is to identify the pins that supply SRAM with power. In most cases, it is impossible to locate a specific pin on the circuit board, because SoC chips use advanced packaging, such as BGA. However, it is not essential to find exact pins in an SoC package as supply pins are connected to passive components (e.g., decoupling capacitors) or circuit-board-level test pads, which tend to be located near the PMIC (§3.5). The layout of passive components follows a typical pattern illustrated in Figure 3.3. For *Volt boot*'s evaluation platforms, I list the test points and target domain's pin names in Table 3.3 and present them in a visual form

Boards	PCB test pads to probe	Nominal voltage	Target memories	Power domains
Raspberry Pi 3	PP58	1.2V	L1D, L1I, registers	Core (VDD_CORE)
Raspberry Pi 4	TP15	0.8V	L1D, L1I, registers	Core (VDD_CORE)
i.MX53	SH13	1.3V	iRAM	Memory ($VDDAL1$)

Table 3.3: Volt boot evaluation platforms, test pads, and their nominal voltages.

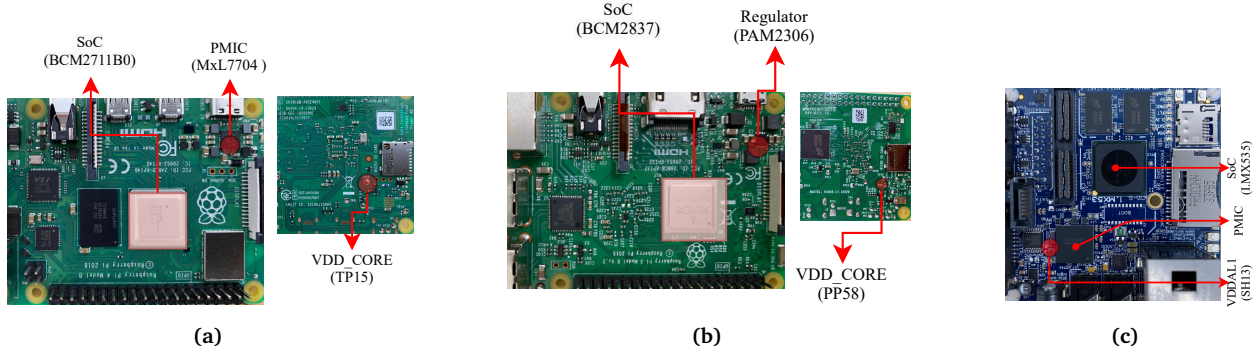


Figure 3.5: Pictures of the evaluation platforms showing the test points where voltage probes are attached. The platforms include (a) Raspberry Pi 4, (b) Raspberry Pi 3, and (c) i.MX535.

in Figure 3.5.

2. **Attaching a voltage probe:** I measure the nominal voltage at the pin(s) and attach an external power supply probe at the same voltage level. The power source needs to supply sufficient current so that the level stays the same when I turn off the device's main power; otherwise, we risk losing the data. As an example, a Raspberry Pi 4 draws current through the test pad TP15 when I attach an 800mV probe. The current varies between 400mA to 600mA depending on software workload. When the SoC's main supply line (powered through a USB C) is disconnected abruptly, the cores draw power from the attached probe. The probe maintains the voltage level even if the cores demand a momentary current surge. The current consumption drops to 8mA after a few microseconds, and the memory domain stays in this retention state indefinitely.

3. **Power cycling and booting the system:** Once the external probe is in place, I disconnect the device from the main power source while the voltage probe keeps the target SRAM active.

The method by which a system powers on and boots after being disconnected from power

can differ. Certain systems enable booting from alternate storage mediums solely after the user's data on the disk has been cleared, while others can initiate booting internally without requiring any external storage media. To replicate this behavior, I configure Raspberry Pis to boot from an alternative medium by using a USB mass storage device. Additionally, I develop a software that runs after reboot to extract data, performing the following operations:

- (A) Reduce contamination on the SRAM's retained data during boot-up by avoiding storing data to it (either explicitly or implicitly).
- (B) Exfiltrate data from the SRAM to other memory (e.g., Flash, DRAM, or a debugger) for post-processing.

The cache extraction software executes CP15 instructions and reads out the *data register interface* of the caches to general-purpose CPU registers. Cache access requires read/write to system registers. For processors with out-of-order execution capability, one must use appropriate data and instruction synchronization barriers. For example, Cortex-A72 processors uses SYS #0, c15, c4, #0, <xt> instruction to execute RAMINDEX operation (cache access request to CP15 co-processor). Data and instruction synchronization barrier instructions DSB SY and ISB, respectively, must follow this instruction before reading the cache data output register interface. A set of general load/store instructions moves the data from the general-purpose CPU registers to DRAM for further processing.

I directly dump the iRAM's through the debug interface, because i.MX535 requires no external firmware support for booting up. Thus, I connect a JTAG probe and directly read out the processor's (Cortex-A8) iRAM contents.

4. **Analysing the memory contents:** Depending on the target SRAM and the objective, an attacker needs to adapt post-processing. Since *Volt boot* reads out the memory without any error, the noise source in a successful attack is the *dynamic behavior* of software and its effects on the data stored in embedded SRAM. For example, error-free key extraction from a cache memory depends on the processing core's workload and other background processes.

3.6.2 How Much Memory is Accessible to an Attacker?

At startup, the CPU uses some types of embedded SRAM before even an attacker has access to those memories. What percentage of memory is available after SoC boot-up depends on the target memory type of an SoC. To find the accessible SRAM, I execute bare-metal software that populates a target memory with predefined patterns. The bare-metal setup allows me to calculate the effect of the CPU's boot phase on internal memories, avoiding dynamic behavior, such as cache eviction. Once the software loads the data/instruction in the target memory, I execute the steps discussed above (Section 3.6.1).

My investigations into the Level 1 (L1) caches on the BCM2711 and BCM2837 SoCs show that they are unaffected by operations during the initial boot process. This observation is consistent with the design of these System-on-Chips (SoCs), where the L1 cache is activated through software control. Hence, unless external software explicitly enables the cache, it remains inactive, allowing an attacker to access a complete snapshot of the L1 cache contents. It's important to note that in these Broadcom models, there's an integrated video core that utilizes the same Level 2 (L2) cache as the ARM CPU cores. During the boot sequence, the video core begins system setup with pre-loaded binaries that overwrite the contents of the L2 cache, thus blocking access to any data following a reboot.

The i.MX535 has similar behavior for caches, but boot ROM uses part of the iRAM as scratchpad memory before initializing the DRAM controllers. That is, the CPU resets part of the iRAM before allowing any debug connection or software execution. Such a boot method is standard among Cortex-M devices; they usually clobber 2KB SRAM (main memory) at the boot phase [71, 114]. I experimentally show that approximately 95% of an i.MX535's iRAM is available to an attacker.³

³A viable defense could hide secrets in the 5% of memory overwritten during the initial boot phase, although I did not verify if this entire portion of SRAM gets overwritten.

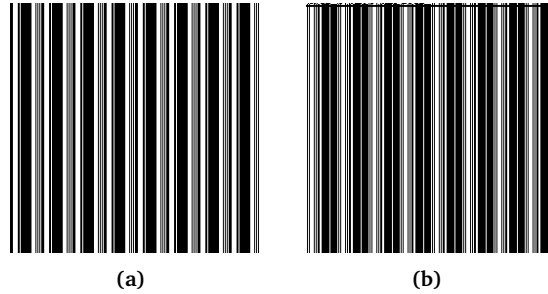


Figure 3.6: Snapshots of i-cache after attacking bare-metal software in (a) BCM2711 and (b) BCM2837 SoCs. Uninitialized cache cells power on into random state (see Figure 3.2), but when I execute *Volt boot* attack, instructions stay in the i-cache across power cycles.

3.7 Attack Execution in Different Memories

A successful cold boot attack depends on many variables, including temperature and an SRAM's intrinsic data retention time (mostly determined by the manufacturing technology). An attacker must turn off the device abruptly to prevent memory corruption or any defensive wiping. The next step is to reduce the temperature of the device; the lower the temperature, the more accurate the extracted information (given some fixed amount of time without power). Then, the victim device needs to boot. The most common way to execute an abrupt power-off in a commercial device is to remove the battery or power connection from the device. These operations require more than a few hundred milliseconds, far too long for SRAM data retention [6].

Volt boot is a non-invasive memory disclosure attack that replaces the *temperature effect* for voltage-induced cross-power-cycle data retention for SRAM. The ultimate result of such data retention resembles a cold-boot-style attack with higher accuracy—without exposing a device to unrealistic low temperatures. *Volt boot* exploits power domain separation in modern SoCs, eliminating variables like data retention time and temperature. I execute three example attacks using the devices listed in Table 3.2. The following proof-of-concept attacks empirically demonstrate the vulnerability of computation methods that store secrets as plaintext in caches, registers, and iRAMs.

3.7.1 Attacking Caches

It is common for System-on-Chips (SoCs) to have large caches. Some on-chip computation methods, such as *CaSE* [190], use a part of these caches as the primary software execution memory instead of external DRAM. However, it is crucial to evaluate the security of such methods under the threat model described in Section 3.4.

Attacking Caches with Bare-Metal Software

In this scenario, I study how *Volt boot* attacks a device that runs bare-metal software and emulate common embedded systems that are designed for specialized applications, such as monitoring and collecting data. I write a bare-metal program that enables the caches, then executes NOP instructions in all four cores. This allows me to quantify precisely how much of the software *Volt boot* extracts. To tightly control the execution environment, I write the software in assembly (i.e., aarch64).

I execute the attack steps (Section 3.6.1) on Raspberry Pis and compare the cache content to the ground truth machine code.⁴ Figure 3.6 provides a visual representation of the cached content after I execute the attack (compare it to Figure 3.2). As expected, the accuracy of data retention in the caches is 100% in all four cores of both devices.

Attacking Caches with an OS

I show how *Volt boot* leaks data from a user application running on a system for general-purpose operations—the Linux kernel. I write an application that stores a specific pattern (0xAA) in a large data structure and reads it back. While executing, I carry out the attack steps from Section 3.6.1 and plot the post-attack snapshots of the respective d-cache in Figure 3.7. The d-cache contains the expected pattern (i.e., 0xAA). To determine if the instructions are present in the cache, I use *grep*

⁴BCM2837 (Cortex-A53) i-cache stores instructions and ECC in each cache line. The order of the bits is undocumented in the technical reference manual [12]. I extract the cached content before and after the attack, which provides an error-free comparison.

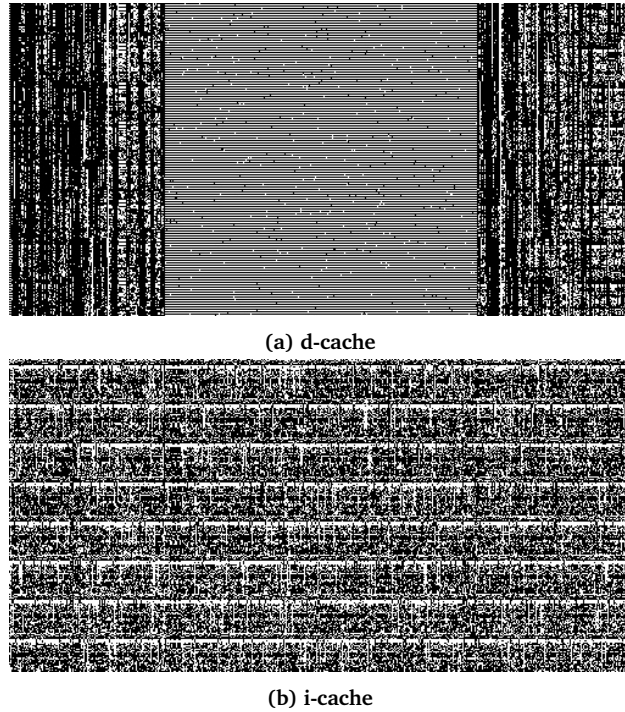


Figure 3.7: Snapshots of the caches after executing *Volt boot* on a system running a general application. I generate the cache images from one WAY of each type of cache.

to search the contents of the instruction cache. This allows me to verify that all the instructions for my software, as compared to the verified machine code (the ground truth), are located within consecutive address spaces.

To quantify the effect of a cache’s dynamic behavior on a Linux-based system, I write a microbenchmark with variable array size; the benchmark loads the array from the Flash to DRAM (and d-cache). I run the benchmark in a standard Raspberry Pi OS running on a Raspberry Pi 4 [47]. This SoC has a 32KB two-way set-associative data cache. I vary the number of 8-byte elements in the array by increasing the size of the array in each set of experiments. The size of the array varies from 12.5% (4KB) of the cache size to full-cache size (32KB); by extension, the number of elements in the array varies from 512 to 2048. I repeat *Volt boot* attack on each array size three times and calculate an average number of elements retrieved for each array size.⁵ I launch one benchmark process per core, which allows me to analyze how L1 cache’s (per core) dynamic behavior affects

⁵That is, a total of 12 experiments for four different array sizes.

	4KB			8KB			16KB			32KB		
	Core 0	Core 1	Core 2	Core 3	Core 0	Core 1	Core 2	Core 3	Core 0	Core 1	Core 2	Core 3
W0	373.0	338.7	354.7	363.0	591.0	580.7	564.7	581.0	1177.3	1155.0	1179.7	1114.3
W1	309.0	341.0	340.7	318.0	633.0	659.7	656.7	656.7	1067.3	1097.3	1084.7	1139.0
W0 $\cup$ W1	512.0	512.0	512.0	512.0	1024.0	1023.7	1024.0	1024.0	2048.0	2048.0	2045.0	2048.0
% data extracted	100.00%	100.00%	100.00%	100.00%	100.00%	99.97%	100.00%	100.00%	100.00%	100.00%	99.85%	100.00%
									91.49%	91.63%	91.78%	85.67%

Table 3.4: Extracted data from d-cache of a BCM2711 SoC using Volt boot attack. The size of the d-cache in this SoC is 32KB, which is divided into two ways, W0 and W1.

Volt boot's data retrieval accuracy.⁶ In each experiment, my post-processing script compares the array elements with the retrieved cache image of each core. Note that, other processes (and the kernel) evict cache lines, therefore, an element of the array can be in both ways of the cache in a modified state. I consider an element of the array present in the d-cache only when the entire 8-byte array element is present in the cache. Table 3.4 lists the results of the L1 data cache data extraction experiment. A 4KB array contains 512 array elements and *Volt boot* retrieves all the elements. *Volt boot* retrieves approximately 90% of the array elements when the array size is close to the cache size. That is, when the data size approaches the total cache size, information retrieval accuracy decreases. The kernel's background processes introduce errors in the data extraction by evicting cache lines when the size of a data structure is comparable to the cache size. Note that in the case of on-chip crypto, which uses cache locking (e.g., *CaSE* [190]), *Volt boot* retrieves the entire binary of plain-text software since neither the kernel nor other processes can evict secret-holding cache lines.

3.7.2 Attacking CPU Registers

Modern SOCs contain different types of CPU registers that are not part of a typical boot sequence, for example, ARM cores use vector registers <v0 . . . v31> to process SIMD and floating-point instructions. These registers are relatively large (128-bit) and byte-addressable, making them suitable for storing security-sensitive states (e.g., key schedules) of cryptographic algorithms, such as AES. Given the threat model, I investigate whether these registers are vulnerable to *Volt boot*.

I write a bare-metal program in aarch64 assembly that fills out the vector registers with distinguishable patterns, e.g., 0xFF and 0xAA. My post-attack analysis on BCM2711 and BCM2837 shows that these vector registers fully retain their states when I execute *Volt boot* attack. Therefore, any on-chip cryptographic program that hides secrets in these registers is vulnerable to *Volt boot*.

⁶At the time of the attack, the victim system concurrently runs four processes in the four different cores of the Cortex-A72 CPU.



Figure 3.8: Visual representation of iRAM's data extraction for address (a) `0xF8000000 to 0xF8007FFF` (b) `0xF8008000 to 0xF800FFFF` (c) `0xF8010000 to 0xF8017FFF` (d) `0xF8018000 to 0xF8020000`.

3.7.3 Attacking iRAMs

iRAMs (also known as OCRAM) are on-chip memories that an SoC uses as temporary storage for different applications, such as boot firmware and multimedia streaming. I study the vulnerability of these memories to *Volt boot* attack using a multimedia SoC, the i.MX535 [123], which contains 128KB of iRAM. This memory is in the L1 memory power domain, and it draws power from the VDDAL1 pin of the SoC. Unlike the BCM2711 and the BCM2837, the i.MX535's ARM core itself draws power through a different pin, VCCGP. The i.MX535 boots from internal ROM, and attacking this SoC does not require any external boot media (e.g., Flash). That is, this device essentially behaves as a microcontroller at startup. I attach a JTAG reader to read/write to the iRAM directly and store four copies of a 512×512 (128KB) bitmap image to quantify the accuracy of data extraction through *Volt boot* attack.

Figure 3.8 illustrates the images that I extract from an iRAM using *Volt boot*. As mentioned before (Section 3.6.2), full iRAM is not retrievable because internal boot firmware partially clobbers it before releasing the core to external software. The overall error in the iRAM's extracted information is 2.7%. To locate the exact error source, I calculate the Hamming distance between the image binary and iRAM extracted binary at a 512-bit granularity (Figure 3.9). The location of the error is clustered around the beginning and end of the iRAM. The largest source of the error address range is between `0x0xF800083C to 0xF80018CC`. The device resets this part of the memory, and the error source is consistent with the other i.MX535 devices.

3.8 Countermeasures

To assess potential countermeasures, I break down the *Volt boot* attack into two broad phases. To execute a successful *Volt boot* attack on an SoC, an attacker must be able to:

1. induce SRAM data retention across power cycles.
2. access the unmodified SRAM contents after the system reboots.

To defend against the attack, one can eliminate any one of these essential steps. This section provides insight into some potential countermeasures to *Volt boot* attack.

- **Eliminating power domain separation:** The decision to separate circuit blocks into power domains involves numerous levels of hardware design stack, ranging *from* device manufacturing *to* architecture. For example, voltage domain separation allows an SoC's power management unit to change a block's behavior when computational demand rises dynamically and to turn it off entirely when not needed. The exposed pins of these domains filter out the noise and maintain stable internal voltage. Eliminating power domain separation is not a practical countermeasure due to performance, efficiency, and implementation concerns.
- **Purging residual memory:** A straightforward way to avoid on-chip data retention in the caches and other on-chip RAMs is to erase the memory as part of the processor's power-down sequence. Such a software/hardware-driven approach is not a practical solution to defend against *Volt boot* because an abrupt power disconnect from a live device stops all operations immediately.
- **Resetting SRAMs at startup:** Even if an attacker successfully retains the memory states after a power cycle, it becomes useless if there is no feasible method to extract the retained information after a reboot. Resetting the memory using hardware such as *MBIST* or other hardware-driven methods prevent this attack.

My experiments on numerous devices, ranging from microcontrollers to application processors, suggest that hardware memory reset at boot-phase is uncommon. Most devices boot into an undefined SRAM state and remain that way until software writes to memory. `armv8.A` suggests that the L2 cache can be reset by pulling `nL2RST` pin low for 16 cycles, but this is not applicable for L1 caches.

When it comes to caches, the CPU does not benefit from resetting the data and instruction RAMs because the cache operation is dependent on the status of tag RAMs (e.g., L1D-tag and L1I-tag), not the data RAMs (e.g., L1D-data and L1I-Data). Therefore, there is no way to reset the data RAMs in the cache other than executing the zeroization operation provided by the ISA, which is a software method. Instead of attempting to reset the memory by writing zero to it, I can adopt a more straightforward hardware-based solution that internally toggles the power of SRAMs at reset. While effective, this solution requires hardware modification, and therefore, is not applicable to already manufactured SoCs.

- **Trust-Zone support:** ARM Trust-zones(TZ) is a hardware-backed memory isolation technique available in most Cortex-A profile processors.⁷ Hardware checks every memory access to assess the validity of the transaction. Cortex-A profile processor stores a security bit (NS) for every line of its cache. Enforcing TZ-support prevents unauthorized access (from a non-secure state) to memory locations that are marked secure. An attempt to access a secure line from a non-secure state triggers a hardware exception. The secure data remains inaccessible to an attacker across power cycles when TZ is enforced because the only way to read secure memory content is to change the security attribute of the memory location; such a reset erases the memory content.
- **Mandated authenticated boot:** *Volt boot* needs to boot the system with an exploitable system image. Signing the system with OEM's signature and burning the hash of the image in the fuses prevents an attacker from booting a device from another media. Note that all devices do not have mandated authenticated boot functionality as it complicates post-

⁷ARM introduced Trust-Zone in Cortex-M profile devices recently.

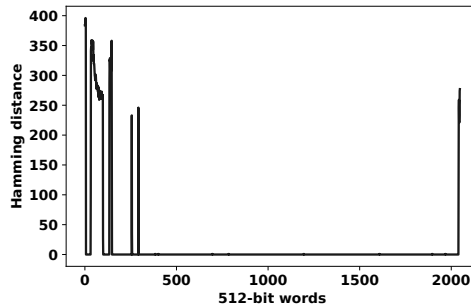


Figure 3.9: Hamming distance between image binary and post-attack binary.

deployment firmware updates. In addition, some processors boot from internal boot ROM using its internal RAM as a scratchpad and allow direct access to on-chip memory because, of the assumption that on-chip SRAM does not store any information across power cycles.

3.9 Related Work

The trend towards ubiquitous computing results in an increased attack surface of devices where attackers have physical access. As society increases its dependence on these exposed devices, device security in the physical access threat model grows in importance and criticality. Realizing this, researchers have begun to focus on physical memory disclosure attacks and defenses.

Cold booting is a well-known attack that affects virtually any device with volatile memory with temperature-dependent data retention. *BootJacker* [26] proposes a reboot attack where an attacker gets physical access to a device and force reboots it to a malicious kernel. The authors construct a proof-of-concept attack where the system recovers its state exploiting the data retention property of DRAM cells. Since the reboot method is software controlled (*i.e.*, warm reboot), this attack is easy to mitigate using an OS-level residual memory purger [34]. Halderman *et al.* [59] proposes a more comprehensive attack on DRAM that exploits DRAM's ability to retain data when the temperature is reduced *well below freezing*. As DRAM's discharge rate is lower at cold temperature compared to the typical operating temperature, it allows an attacker to remove the DRAM from a victim

machine and plug it in another machine. The attack reconstructs security-critical information such as disk encryption keys from a cold-boot-extracted memory image. Even though DRAM's discharge rate reduces significantly at low temperature, some bits do flip states during migration between the victim's and attacker's machine. These bit flips introduce errors in the extracted information. Consequently, the attacker needs to perform a time-consuming search to reconstruct the correct keys from the recovered content; the search space grows exponentially with the number of flipped bits. In follow-up work, researchers reproduce the cold boot attacks with more modern DRAM technology [18, 94].

DRAM in mobile and IoT devices is usually soldered on the circuit board, which impedes the conventional plug-and-play cold boot attack. FROST [119] shows how an attacker erases an android's non-volatile memory by triggering a factory reset (without erasing the DRAM). Once the user data is erased from a device, an attacker loads a lightweight kernel to dump and analyze the victim's cell phone contents.

Modern systems make it challenging for an attacker to extract sensitive information from post-reboot DRAM memory dump using a low-temperature effect. The defense mechanisms to DRAM cold boot attacks can be broadly classified into two sets. One set of methods scrambles or encrypts the outgoing data. For example, DDR4 uses session keys to scramble data in a DRAM to prevent cold boot attack [186]. Another set of methods advocates overwriting memory contents using built-in hardware every time a DRAM chip powers on [125, 151]. Researchers have evaded memory scrambling in more recent cold boot attacks [186] and self-resetting DRAM has not been implemented or deployed by commercial DRAM vendors.

3.10 Conclusion

The last two decades of hardware security research has seen a rampant increase in proof-of-concept *and* real-world attacks targeting *off-chip memories* designs. Recent efforts to mitigate these attack

surfaces primarily turn to *on-chip* computation—i.e., *Cache as RAM*[190]—as their deeply embedded nature renders all previous classes of attacks (e.g., cold boot[59]) obsolete. However, these systems’ reliance on *power domain separation* enables a *new* class of attacks: *Volt boot*.

In this work, I show that current on-chip SRAM is indeed resilient against conventional temperature "freezing"-based attacks (e.g., [6]). However, I show the effectiveness of a voltage-based attack that snapshots SRAM, *without* exposing an SoC to low temperature, effectively enabling the indefinite retention of SRAM data, while software changes. Compared to previous-generation cold boot attacks against standalone SRAM, *Volt boot* achieves *error-free* data exfiltration on devices spanning *three* distinct microarchitectures—defeating the paradigm of on-chip computation as a viable defense against secret extraction.

Chapter 4

Hiding Secret Messages in SRAM's Analog Domain

4.1 Introduction

In many cases, encryption alone is insufficient as revealing the transmission of secret messages opens one to the threat of physical coercion (commonly called a rubber hose attack [20]) or even death [86]. Historically such ramifications were limited to spies, who used dead drops as a cover for communication. For example, during the cold war, Martha Peterson used dead rats as literal dead drops to communicate with Aleksandr Ogorodnik, a member of the Soviet Foreign Ministry, who was relaying secret foreign cables [171]. The proliferation of computing devices and the rise of concern about information passing across national borders push the need to hide the presence of secret messages down to the level of normal citizens. Consider that at the United States border alone, searches of electronic devices increased 400% over the last five years [172, 173]. Such searches extend well beyond the borders of the United States, with potentially worse consequences in other nations [14, 39, 53]. In response, travelers are already looking for ways to hide information. For example, in 2012, a Syrian refugee smuggled evidence of human rights violations by hiding memory chips in a wound [17, 118]. Given that border controls are getting tighter, with greater emphasis on electronic devices, and more advanced analysis [102], the need to hide the act of communication grows.

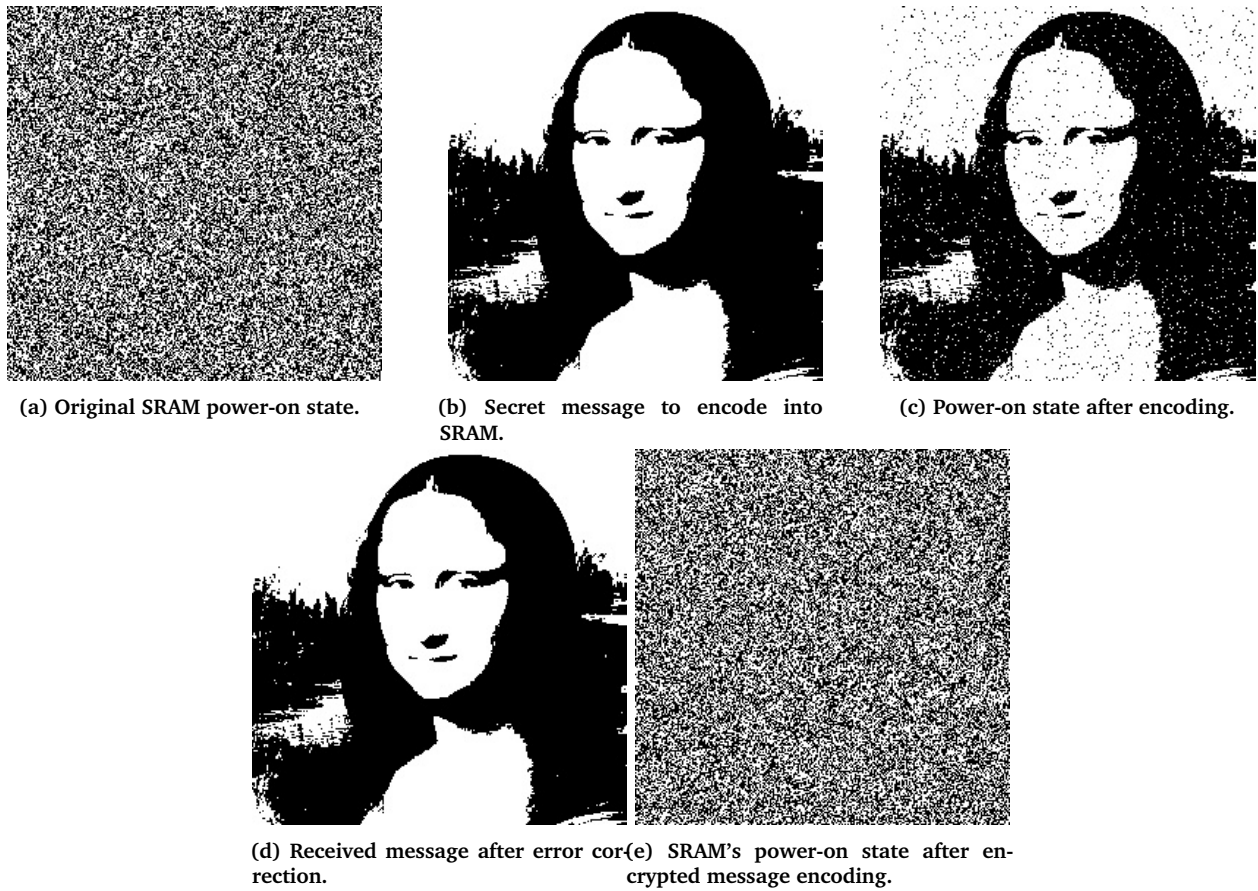


Figure 4.1: *Invisible Bits* and its various use cases on SRAM embedded in Texas Instruments MSP432 [71]

As restrictions continue to reduce the free flow of information, hiding the presence of secret messages becomes paramount; hiding the existence of a secret message prevents forceful key disclosure or other attacks from being attempted, because the attacker is unaware of the communication. Such message hiding is referred to as steganography. Steganography is communication over a covert channel that is modulated on top of everyday objects. A key property of steganography is plausible deniability: attackers are unable to distinguish between objects that carry covert information and objects that do not. Plausible deniability is what allows for covert communication in plain sight. An effective steganography system provides both analog- and digital-domain plausible deniability and is able to withstand normal use of the carrier object without disturbing or revealing the secret message modulated on top.

Given the value of being able to hide the existence of data, researchers propose a range of steno-graphic techniques that work at various levels of the system stack. The most popular techniques work at the system level, hiding data in images [42, 178], audio files [185], video files [153], and within file system metadata [19, 111, 127]. Another set of techniques hides data in the transmis-sion of cover data across a network [187]. These approaches tend to be very low capacity, because they cannot interfere with the cover data—otherwise, plausible deniability is lost. A higher capacity set of techniques encodes data in the device memory itself as opposed to in digital files stored by the memory; this set of techniques that hides data in the analog domain of a device is referred to as on-chip stenography. The main target of existing on-chip steganography techniques is Flash memory: (1) the charge held in a Flash cell [193] and (2) the program/erase time of a Flash cell [177]. While these approaches marginally increase the capacity of file- and network-based techniques, they **all suffer from limited resilience, i.e., it is easy for the attacker to erase the message, even if they cannot prove one exists.**

In this work, I design and implement an on-chip steganographic technique that uses a more ubiq-uitous on-chip memory that dramatically increases capacity and resilience—while providing both digital- and analog-domain plausible deniability. *Invisible Bits*, encodes secret messages into SRAM's analog-domain properties by controlling and accelerating the natural aging process that all transis-tors experience. SRAM's power-on state is unique in that individual cells power-on to a value deter-mined by the relative analog-domain properties of the cross-coupled inverter gates at the heart of every cell. The value an SRAM cell holds determines which of the two inverters is active. While an inverter is active, it gradually ages, changing its analog-domain properties; given enough aging, the power-on state of the cell will change, making analog-domain changes visible in the digital domain through its power-on state. I show how to control the direction and rate of this aging process in a way that allows us to dictate the power-on state of >90% of SRAM cells. I show that even with such a significant change in the analog domain, the device functions normally at the digital level. Figure 4.1 shows an example image encoded into a device using *Invisible Bits*.

Invisible Bits has the following advantages as a steganographic technique:

- **Covert:** *Invisible Bits* does not store information in a traditional sense, which is attributable to the volatility of SRAMs. The *payload* is decoupled from the normal functionality of SRAM as memory, which provides digital-domain plausible deniability. To protect against more powerful adversaries, *Invisible Bits* encrypts the payload before encoding, which ensures the indistinguishability of the power-on states of devices with a message encoded and those without.
- **Erase/Write tolerant:** *Invisible Bits* is robust against general-purpose device operation. That is, an encoded SRAM withstands regular memory operation (*i.e.*, read/write) without introducing significant error in the hidden message.
- **Ubiquity:** Most computing devices use on-chip SRAM either as a cache (e.g., desktop-class devices), or main memory (e.g., embedded devices), or configuration memory (e.g., Field-Programmable Gate Arrays), and so *Invisible Bits* applies to all of these devices.¹
- **Constant time operation:** Since SRAM cells age due to the feedback loop inherent in their structure, all SRAM cells (that have power) age concurrently. Given the speed of writing to SRAM, payload encoding time is largely independent of SRAM size and payload size, but dominated by the time it takes to change SRAM's analog-domain properties so that they affect its power-on state.
- **Copy tolerant:** Sampling power-on state at normal operating conditions does not alter the encoded payload.

This work makes the following contributions:

- I show that directed SRAM aging is a covert channel that one can exploit to transfer data with plausible deniability (§4.4).

¹*Invisible Bits* requires the ability to access SRAM power-on state for a given device, before it is overwritten.

- I implement and evaluate an end-to-end steganography system using modern, commercially available computing devices (§4.6).
- I improve information capacity by 100x compared to Flash-based on-chip steganographic techniques (§4.5.3).
- I show that the proposed system is resilient to long-term natural recovery (§4.5.1) and that it retains the encoded message reliably under normal operating conditions (§4.5.1).
- I implement and evaluate both error correction and encryption on top of *Invisible Bits*, showing that encryption provides analog-domain plausible deniability (§4.5).

4.2 Background

As we have seen in chapter 2 SRAM stores data in a positive feedback loop between two inverters. When we do not write anything to a cell, the logic states it powers on at is determined by the hardware race condition at startup.² The grey curves in Figure 4.2b show that after $2ns$ of powering the cell up, node A and B settle at V_{dd} and ground, respectively. Because the threshold voltage of M4, $|v_{th4}|$, is less the threshold voltage of M2, $|v_{th2}|$, M4 switches on before M2 and eventually pulls node A up to V_{dd} . Node A is connected to the gate of M1, and it enters the saturation region (from cut-off) and pulls down node B to the ground. Thus, the cell’s power-on state is 1.

4.2.1 Influencing Power-on Behavior

Circuit aging is a natural phenomenon that gradually *increases* the threshold voltage of transistors [16]. Bias Temperature Instability (BTI) and Hot Carrier Injection (HCI) are the primary causes of circuit aging in modern technology nodes [145]. For the purposes of analog-domain changes revealed through power-on states, BTI is of interest, while HCI has less effect [145] and affects both

²I reproduce the schematic diagram (Fig. 4.2a) from chapter 2 for convenience.

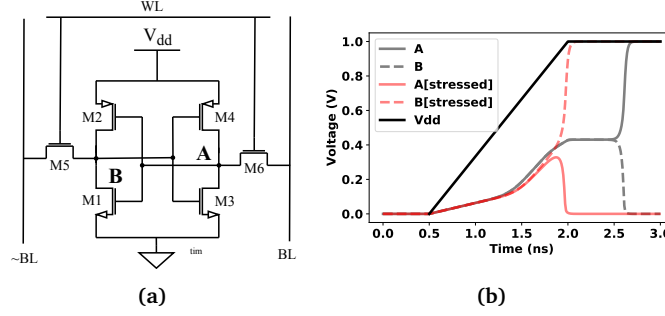


Figure 4.2: (a) A schematic diagram of an SRAM cell (b) simulation waveform of startup state of the cell.

inverters equally since HCI involves switching and both inverters switch together. MOS devices suffer from BTI because of surface state generation in the oxide and substrate interface [60]. Positive Bias Temperature Instability (PBTI) affects the NMOS, while Negative Bias Temperature Instability (NBTI) affects the PMOS [75]. When a PMOS is operational (*i.e.*, $V_{sg} > |V_{th}|$), a strong vertical electric field creates interface states, which increases the threshold voltage over time. When $V_{sg} \leq |V_{th}|$, the NBTI effect recovers, it partially brings back the threshold voltage closer to nominal. Note that the recovery is partial, and so the magnitude of the threshold voltage increases over time. A similar effect is observable in NMOS devices [113].

Aging-induced degradation has the ability to determine the power-on state of SRAM cells.³ The power-on state of a cell depends on the post-manufacturing variation in transistors and the magnitude of aging the transistors have experienced up to this point; the value held by the cell determines which transistor is aged. This means that SRAM cells undergo data-directed aging. I illustrate data-directed aging using a MOSFET Reliability Simulation (MOSRA) in HSpice [170]. Figure 4.2b plots waveforms generated from pre-aging and post-aging transient simulation using 45nm predictive technology model [15]. Assume the cell in Figure 4.2a is initially biased towards 1. Considering only PMOS mismatch, M4 turns on earlier than M2; hence, node A is pulled up to V_{dd} while node B is pulled down to the ground. If the cell is kept operational in this state (with node A = 1 and node

³The impact of transistor aging is known to some degree by the SRAM PUF community, as modest aging has been used as a denial-of-service attack on SRAM PUFs [141]. Though it is not the focus of this paper, the results of our extreme/controlled aging suggest that it is possible to clone SRAM PUFs.

$B = 0$), M4’s threshold voltage gradually increases (due to NBTI aging). The increased threshold voltage causes M4 to turn on at a later time during the next power-on event. Given enough aging time, the cell ultimately becomes biased towards 0 as M4 now turns on later than M2. The red waveforms in Figure 4.2b capture this change.

I run two experiments to demonstrate the concept of (1) software-directed and (2) accelerated aging. As discussed, stressing a cell with a logic state biases its power-on state complement of that logic state. That is, I expect the number of cells that power-on to 1 to increase if I write 0 to the entire SRAM and then aggressively age the device (Figure 5.7a). Conversely, I expect increased 0s if I fill SRAM with 1s before aging (Figure 4.3c). Compare these figures to the unaged SRAM in Figure 4.3a.

In the second experiment, I demonstrate the knobs that *Invisible Bits* exploits to accelerate NBTI aging for the purposes of encoding messages in SRAM’s analog domain. Previous work shows that both supply voltage and operating temperature of a device affect the rate of NBTI aging. To validate this, I write 1 to all cells of four unaged MSP432 microcontrollers. I then exposed each microcontroller to a different combination of voltage and temperature, with nominal conditions being (1.2V, 25°C) and accelerated being (3.3V, 85°C). Figure 4.3d shows that voltage has the largest acceleration effect, which is magnified by increased temperature, a result supported by previous work [101].

4.3 Threat Model

After turning on a device, the SRAM embedded within the device retains its power-on state until software overwrites it. SRAM’s uninitialized power-on state possesses interesting properties such as temporal and spatial randomness, making it an attractive security primitive in numerous applications such as physical unclonable functions (PUF), random number (TRNG), and device fingerprint generators [61, 63, 143]. As long as SRAM stores some data, regardless of whether it is from a

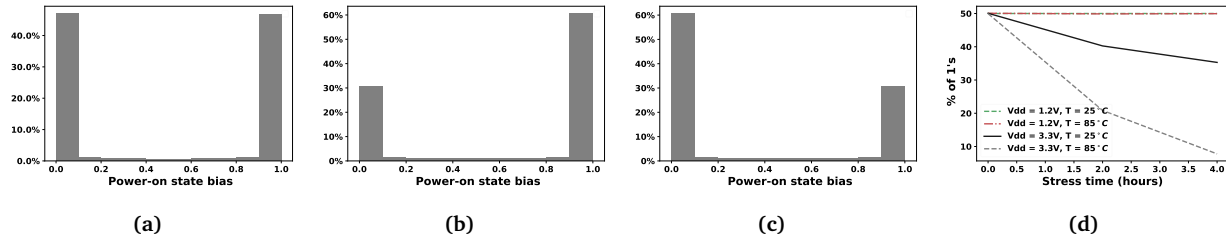


Figure 4.3: Illustrates (a) startup distribution of an SRAM, data directed aging (b) when an SRAM holds logic 0, (c) logic 1 during stress. (d) Illustrates accelerated aging using supply voltage and operating temperature.

power-on event or from software, it undergoes analog-domain changes in a process called aging; such analog-domain changes are reflected digitally in future power-on states [81]. *Volt boot* harnesses, directs, and accelerates SRAM aging as means to reliably encode secret messages—without altering the functionality of the SRAM.

I adopt a threat model similar to Flash-based on-chip steganography techniques [177, 193]. In this threat model, the goal is for two parties to communicate via a plausibly-deniable covert channel; depending on the use case, the parties may be separate entities (e.g., for data transmission), or the same entity separated in time (e.g., for data storage). The communicating parties have full control over selecting a device as a steganographic medium. The threat model also assumes that the parties have a pre-shared key that they use for message encryption/decryption. Lastly, in the on-chip threat model, data transmission is the limiting factor, not throughput, as encoding and decoding are separated by the time it takes to transfer the device; capacity remains crucial as it determines if a message can fit on a single device.

The threat model assumes that the adversary gets temporary access to a device, otherwise, the attack devolves into a denial-of-service attack, which is impossible to stop. While in the adversary's possession, they can inspect, copy, overwrite, and erase its digital contents. In addition to data manipulation, the adversary may use the device to validate its functionality (e.g., scroll through pictures and contacts). The only restriction that I place on the adversary is that their analysis is non-invasive and non-destructive.

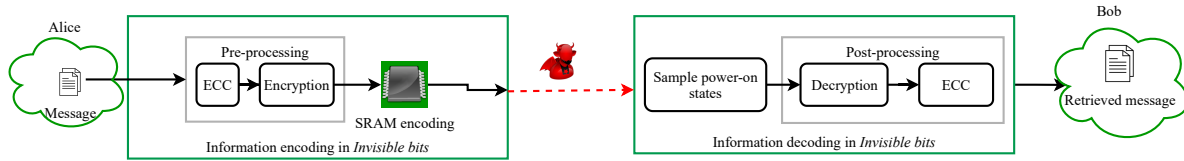


Figure 4.4: *Invisible Bits* overview.

4.4 Design

I leverage the stress-induced analog-domain change of an SRAM to design a plausibly deniable and resilient information encoding scheme: *Invisible Bits*. The design of *Invisible Bits* consists of two major parts: information encoding and information decoding. Figure 4.4 shows the steps involved with sending and receiving a message using *Invisible Bits*. The covert communication process starts by pre-processing the message. Pre-processing is where the transmitting party (i.e., Alice in Figure 4.4) applies their desired error correction and encryption schemes to the original message. The result is the *payload*. The transmitting party takes the payload and creates a program from it; the program writes the payload to SRAM and then halts execution by busy waiting. With SRAM set to the desired state, the transmitting party encodes the state in SRAM's analog domain by stressing the device with increased voltage and temperature. After sufficient stress time (I evaluate the impact of stress knobs in §4.5), the transmitting party passes the device to the receiving party. Examples of this could be through the mail, via a dead drop, or by crossing a monitored border.

Once the receiving party (i.e., Bob in Figure 4.4) has possession of the device, they need to extract the information encoded in SRAM's analog domain. For this, they first load a program crafted to retain SRAM's power-on state, then they capture several power-on states from the SRAM; this process can either be automated, as I do it for my experiments or by hand, since it only requires temporarily removing and replacing the power. The recovered payload is then passed through decryption and error correction steps if they were used during message encoding.⁴

The information encoding scheme consists of two major parts:

⁴I assume that the presence and order of error correction and encryption information are pre-shared between communicating parties.

4.4.1 Message Pre-Processing

Before encoding a message into SRAM's analog domain, it passes through two optional pre-processing steps: (1) where an Error Correcting Code (ECC) is added on top of the message and (2) where the message is encrypted. As shown in the evaluation (§4.5), not all SRAM cells will undergo sufficient aging to change their power-on state to reflect the value loaded during the payload encoding step. Thus, to reduce error beyond what is possible through the stress and time knobs that *Invisible Bits* provides, I make it possible to layer ECC on top of *Invisible Bits*. The actual ECC method is orthogonal to *Invisible Bits*, but my analysis of errors from payload encoding and aging recovery in Section 4.5.2 provides guidance to users as to the most appropriate error correction approaches and the expected error rate and channel capacity.

I also allow users to layer encryption on top of *Invisible Bits*. The obvious benefit of encryption is providing confidentiality, but my analysis in Section 4.6 shows that encryption also provides analog-domain plausible deniability, because SRAM's power-on state properties before payload encoding match that of a random function. For a visual example of this effect, compare the pre-encoding power-on state in Figure 4.1a, with the post-encoding power-on states for a raw message in Figure 4.1c and an encrypted message in Figure 4.1e. Unlike ECC, I note that the choice of the encryption algorithm is crucial: the diffusion that is an important property for block ciphers means that even a single error in a block results in roughly half of the bits of the block flipping—dramatically increasing error rate. For example, using the industry-standard cipher AES-CBC turns an error rate of 0.8% into an error rate of 50% as the first erroneous bit causes the output of all subsequent blocks to become random. Thus, I advocate a stream cipher, which is error-neutral, i.e., error bits in the ciphertext are exactly the error bits in the plaintext, no less, no more. My implementation uses AES-CTR as the stream cipher, where the nonce is the manufacturer's device ID.⁵ Once passed through any error correction and encryption steps, the message becomes a *payload* that *Invisible*

⁵Having per-device nonces ensures that even the same messages produce different payloads. This protects against attacks that look for common prefixes in the ciphertext and attacks that look for the same power-on state across devices.

Bits encodes in the SRAM.

4.4.2 SRAM Analog-Domain Payload Encoding

In this step, *Invisible Bits* encodes the payload into the analog-domain properties of the SRAM embedded in a computing device. The first step is the creation of a software program that writes the payload to the SRAM. To automatically create such a program, I write a tool that takes a payload expressed as a binary file, and returns an assembly program that writes that payload to the SRAM. After the program initializes SRAM's state, it busy waits in an infinite loop. The instructions in the assembly program run from non-volatile memory on the device, i.e., not the SRAM. *Invisible Bits* then assembles this program and loads it onto the target device using the debugger. The second step powers on the target board at nominal conditions to allow the program to initialize the SRAM state. The third step is where the encoding occurs as I increase temperature and voltage to accelerate SRAM aging. After the user-defined stress time, encoding completes, the device is removed from the thermal chamber, and a camouflage program is loaded onto the device.

Algorithm 1 summarizes the entire message encoding process using *Invisible Bits*. First, I apply an error correction code (ECC) on the data (Line 1) and use a pre-shared key to encrypt the data along with the parity bits (Line 2). Then, I load the program to the device running at nominal voltage and temperature (Line 3-4). The physical encoding process starts at Line 5, where I elevate the voltage and temperature to $V_{acc.}$ and $T_{acc.}$, respectively. *wait(stress time)* controls how long I stress the device. Voltage $V_{acc.}$, temperature $T_{acc.}$, and stress time depend on the required bit error rate, as discussed in §4.5.

4.4.3 Decoding Payloads via SRAM Power-on State

Once the receiving party gets access to the device, they have to extract the payload that the sender encoded in SRAM's analog domain. For this, *Invisible Bits* leverages the observation that the power-

Algorithm 1 *Invisible Bits* message encoding

Input: Message (d)**Output:** Encoded computing device*Message pre-processing*

- 1: Apply ECC on the message
 - 2: Apply encryption on the message and parity
- SRAM analog-domain payload encoding*
- 3: Set $\{V_{dd}, T_{emp}\} = \{V_{nom.}, T_{nom.}\}$
 - 4: Load binaries (Instructions and payload)
 - 5: Set $\{V_{dd}, T_{emp}\} = \{V_{acc.}, T_{acc.}\}$
 - 6: wait(stress time)
-

on state of SRAM reflects coarse-grain analog-domain information about each SRAM cell; specifically, the power-on state reveals which inverter gate at the heart of the SRAM cell turns-on first. The goal of the encoding process is to force one of the inverters to turn-on first by slowing the other inverter through accelerated NBTI aging. Note that this means that the payload revealed through the power-on state is actually the complement of the payload used during the encoding step (like a negative in photography). My experiments suggest dealing with inverted encoding requires no special treatment other than complementing power-on state at the decoding phase.

Before capturing the power-on state, the receiver loads a program on the device that ensures the integrity of SRAM's power-on state. For my implementation, I use a program that boots to an infinite loop, that runs entirely out of Flash memory. Then, to extract a reliable representation of the power-on state, the receiver captures multiple power-on states from the device. They use a majority voting scheme on the state captures to produce the retrieved payload. While any odd number of state captures works, I find that taking five captures is sufficient to filter noise.

4.4.4 Decoded Payload Post-Processing

Post-processing in the decoding phase is the same as encoding phases but in the reverse order. The only difference is that I have to start by inverting the majority power-on state from the decoding step. Algorithm 2 lists the steps to decode the information hidden in the power-on states of an

SRAM. The process starts with power cycling the device N times and reading out the SRAM power-on states (Line 1-5), where N is an odd number. In Line 6, *Invisible Bits* apply majority voting on the SRAM states across trials. Once I have SRAM power-on state, I use the decryption and error correction and retrieve the message (Line 7).

Algorithm 2 *Invisible Bits* message decoding

Input: Encoded device

Output: Decoded message

Decoding payloads via SRAM power-on state

1: Set $\{V_{dd}, Temp\} = \{V_{nom.}, T_{nom.}\}$

2: **repeat**

3: Power cycle the device

4: Get power-on states

5: **until** N times

Decoded payload post-processing

6: Apply majority voting

7: Apply decryption and ECC

4.5 Evaluation

Table 4.1 lists the devices I test for the implementation feasibility of *Invisible Bits*. The list contains devices from single-cycle microcontrollers to complex general-purpose processors from different vendors.⁶ I test these devices for their power-on state accessibility and aging acceleration. Although the magnitude of $V_{acc.}$, $T_{acc.}$, stress time, and message extraction error rate vary based on a device's manufacturing technology, SRAMs across the spectrum of devices behave regularly and predictably. Unless otherwise stated, I use MSP432P401 [70] devices for most of my experiments, keeping the analysis and discussion consistent.

I develop a control board that allows us to expedite the evaluation process by automating the power-on state sampling and accelerated aging. Although my setup is automated, encoding and

⁶Most of my tested devices are from ARM as these are popular in mobile devices and provide a standard debug port to access internal memories across different architectures. However, I test devices from different silicon vendors as SRAM aging responses vary depending on the manufacturing technology.

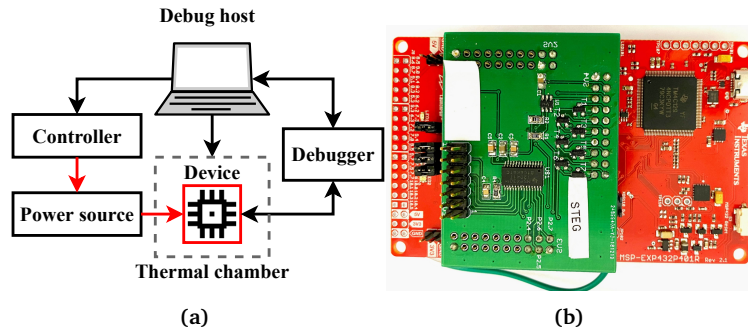


Figure 4.5: (a) Schematic diagram of the evaluation setup. (b) Evaluation platform. The red board is an MSP432P401 launchpad and the green board is my custom control board.

decoding are able to be done by hand without changing the results. Figure 4.5a shows the high-level schematic diagram, and Figure 4.5b is a picture of the evaluation platform. The setup mainly consists of a controller, debug hardware, debug host, power source, and thermal chamber. The controller supplies power directly if the target device consumes a small amount of power (a few mA) but switches to an external power supply unit if the target demands higher current (*e.g.*, processor). The target device runs software written in assembly to read and write to the SRAM. When the target device is a microcontroller, a debugger directly reads out its main memory's power-on state and sends it to the debug host (*e.g.*, PC). However, processor cache access requires co-processor (for ARM Cortex-A device) operations to read cache contents.

All of my measurements eliminate the SRAM data remanence effect by driving the supply voltage of the device to the ground state rather than waiting for it to discharge naturally. my evaluation setup uses a Test-Equity thermal chamber [166] to control the temperature during measurements. To achieve maximum acceleration, I elevate the supply voltage and temperature to 3.3V (2.75x the nominal) and 85°C (3.4x the nominal),⁷ respectively during the encoding period.

⁷I use the device datasheet to guide my search for the stress voltage and temperature.

Devices	CPU core	On-chip SRAM size	On-chip Flash size	Access to power-on state	Accelerated aging	Manufacturer
MSP430G2553 [71]	MSP430 single cycle	0.5KB	16KB	✓	✓	Texas Instruments
MSP432P401 [69]	ARM Cortex-M4	64KB	256KB	✓	✓	Texas Instruments
EFM32WG990F256 [156]	ARM Cortex-M4	32KB	256KB	✓	✓	Silicon Labs
ATSAML11E16A [114]	ARM Cortex-M23	16KB	64KB	✓	✓	Microchip
M263KIAAE [37]	ARM Cortex-M23	96KB	512KB	✓	✓	Nuvoton
M2351SFSIAAP [35]	ARM Cortex-M23	96KB	512KB	✓	✓	Nuvoton
M252KG6AE [36]	ARM Cortex-M23	32KB	256KB	✓	✓	Nuvoton
M251SD2AE [36]	ARM Cortex-M23	12KB	64KB	✓	✓	Nuvoton
R7FS1JA783A01CFM [44]	ARM Cortex-M23	32KB	256KB	✓	✓	Renesas Electronics
STM32L562 [163]	ARM Cortex-M33	40KB	256KB	✓	✓	STMicroelectronics
LPC55S69JBD100 [122]	Dual-core ARM Cortex-M33	320KB	640KB	✓	✓	NXP Semiconductors
BCM2837 (Rpi3) [134]	Quad-core ARM Cortex-A53	L1:256KB, L2:512KB	0KB	✓	✓	Broadcom

Table 4.1: A list of tested devices.

4.5.1 Error Profiling

To exchange or store information reliably, we need an understanding of the system’s error profile. The spatial distribution of error and capacity are the determinants in managing errors in a system. In this section, I analyze the origins and distribution of the error in *Invisible Bits*.

Primary error source

Failure to direct cells’ power-on states, natural recovery and standard device operation are potential error sources in *Invisible Bits*. The error magnitude in *Invisible Bits* relies on the degree to which SRAM power-on states change in response to stress-induced analog-domain degradation. Symmetric and asymmetric SRAM cells respond to stress differently. Stress tends to quickly impact the power-on state of symmetric cells because they are not strongly biased towards any logic state. These cells have inverters with similar analog-domain properties, and it is easier to increase relative differences between them by exposing the cell to stress. On the other hand, variance in the manufacturing process makes some cells extremely biased towards a logic state, limiting my ability to direct the power-on state in the opposite direction. The manufacturing mismatch between the inverters can be so large that stress-induced degradation fails to overcome such bias and leaves the power-on state unchanged after encoding; this is the main error source.

I elaborate on this with an example using Figure 4.2a. Let us assume that $|v_{th2}| > |v_{th4}|$. If all

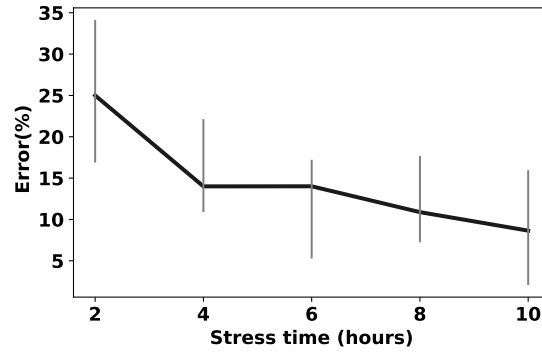


Figure 4.6: Influence of stress time on error.

other parameters (e.g., aspect ratio, doping density, etc.) are the same for M2 and M4, the cell powers up with 1 because M4 wins the hardware race condition. Suppose the cell is stressed with this logic state (node A = 1) for a long time. In that case, we expect the cell to eventually flip in a future power-on state. This implies that there is a flip in the threshold voltage inequality (i.e., $|v_{th2}| < |v_{th4}|$) or at least the gap is reduced to a reasonable range so that across trials, the majority power-on state is 0. Although most cells respond to such stress, a few cells keep their power-on state unchanged even after exposure to stress for a long period. As a result, we cannot direct power-on states of all the cells to my desired direction because of extreme manufacturing biases (e.g., $|v_{th2}| \gg |v_{th4}|$), which fundamentally sets the error floor of *Invisible Bits*.

Stress conditions and duration decide the magnitude of error in *Invisible Bits*. In Figure 4.6, I plot the mean errors of five devices along with maximum and minimum. I keep the stress condition the same for each of the devices to study the effect of stress time on the error magnitudes. As shown in the figure, the error rate follows a logarithmic relation with time; therefore, achieving lower error requires exponentially longer time.

Spatial distribution

The distribution of errors decides what type of error correction is needed in the system to get the desired accuracy. I calculate *Moran's I* to study the spatial distribution of errors, which allows us

to determine whether the error is randomly distributed or bursty. A *Moran's I* statistic close to zero indicates that error is spatially random and a value further away from 0 indicates non-random patterns, closer to 1.0 indicates a positive correlation, and closer to -1.0 indicates a negative correlation.

To quantify the spatial distribution of errors, I write all-1s to one SRAM and all-0s to another SRAM, then expose them to accelerated aging. Table 4.2 lists the spatial autocorrelation of unstressed and stressed SRAMs. For the stressed condition, the autocorrelation is of errors, since I write a single value to all of SRAM. The results of this analysis say that the location of errors is nearly randomly distributed as the autocorrelation approaches the expected value of $-1/(N - 1)$, where N is the number of error locations.

Stress condition	SRAM	Spatial autocorrelation	p-value
Unstressed	1	0.011	0.00
	2	0.009	0.00
Stressed	1 (Stress logic = 1)	0.005	0.00
	2 (Stress logic = 0)	0.004	0.00

Table 4.2: Spatial autocorrelation of power-on states of two SRAMs before and after stress. We stress each SRAM with one logic value (0/1); therefore, the post-stress power-on state reflects spatially-correlated errors in encoding.

Effect of natural recovery

Stress-induced analog-domain changes partially recover, which increases errors in the retrieved payload. I study the effect of natural recovery when a device is inactive during covert communication, assuming that the message needs to stay in the analog domain for a long time.

Most of the transistors in a circuit retain their stress-induced degradation of analog-domain properties [154]. Some transistors, however, partially recover when stress is released [192], which eventually affects the extraction accuracy of *Invisible Bits*. To study the effect of natural recovery on the error, I shelve an encoded SRAM and sample the power-on state at 7-day intervals. The results, shown in Figure 4.7, show that recovery follows a logarithmic relation with time [75], which gets

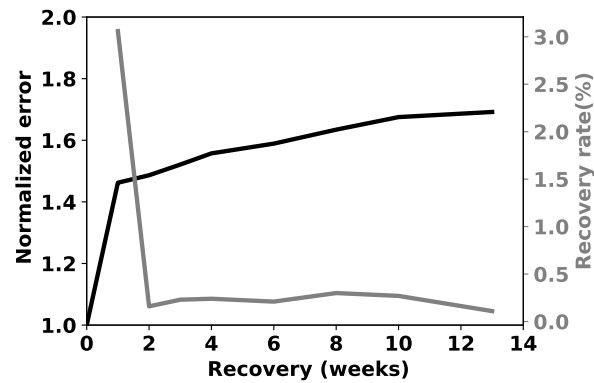


Figure 4.7: Normalized errors in *Invisible Bits* when stress-induced degradation recovers naturally.

reflected in *Invisible Bits*'s error rate. I plot the change in error rate as well to show that the recovery rate decays exponentially with time. The error increases $\approx 1.6\times$ times after one month, which still keeps the error within 10%.

Effect of normal operation

The threat model allows the regular operation of a device without restriction. To evaluate the effect of regular device operation, I design software that continuously writes values to SRAM. To ensure fairness and all possible software, I write a pseudo-random sequence of words to SRAM. The pseudo-random number generator uses a 32-bit linear feedback shift register tailed by a linear congruential generator (from *glibc*, $x_{n+1} = 1103515245 \times X_n + 12345 \bmod 2^{31}$) as seed generator to avoid repetition of numbers in my long-running experiment. I shelve an encoded device operating at nominal conditions (1.2V and room temperature) for a week. I observe a $\approx 1.2\times$ increase in the error after a week-long recovery, which is less than what I observe in the natural recovery experiment ($\approx 1.4\times$). While this seems counter-intuitive, it makes sense when you consider that half the time, the original stress value is being reinforced. Additionally, there is no risk of significantly stressing the opposing inverter because regular operation for a short time period does not significantly impact age transistors (see Figure 4.3d).

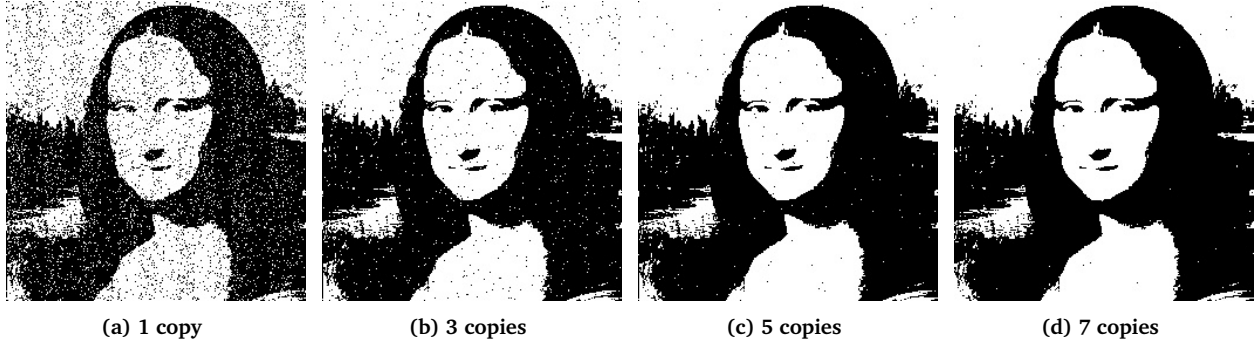


Figure 4.8: Visual example of how a repetition code removes error from the decoded message.

4.5.2 Reducing Error

Invisible Bits allows users to layer on an error-correcting code to further reduce error beyond what is feasible with encoding and recovery. There is a rich history of error-correcting codes, many targeting specific error patterns. To provide guidance on the most suitable ECC algorithms, I analyze the error rate and patterns from earlier experiments.

my analysis shows that the errors in *Invisible Bits* are largely randomly located. Randomly distributed errors are the worst-case for error correction. Depending on error rate, the conventional approaches for dealing with randomly distributed errors are a repetition code, for high error rates, and Hamming codes, for low error rates.

Since my mean error is 10% (§4.5.1), I must start with a repetition code. A repetition code encodes several copies of the payload in SRAM and uses majority voting to determine a representative payload. Having randomly distributed errors allows us to estimate the effect as repetition code Bernoulli trials. Assuming the probability of success is p , n trials reduces the error according to Equation 4.1. For example, 10% error becomes 2.8% when three copies are encoded.

$$Error = 1 - \sum_{i=\frac{(n+1)}{2}}^n \binom{n}{i} \times p^i \times (1-p)^{n-i} \quad (4.1)$$

The error performance of *Invisible Bits* increases logarithmically as I increase the number of copies.

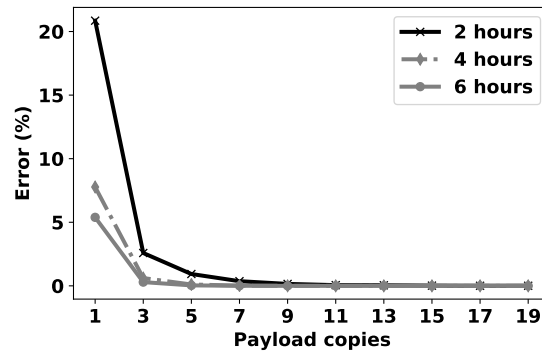


Figure 4.9: Error reduction due to changing stress time and the number of copies used in the repetition code.

To quantify this effect, Figure 4.9 shows how I explore the effect of stress time and the number of copies as means for error reduction. I encode a payload into an SRAM at three two-hour-long stress cycles. The payload is replicated into many copies so that I can use additional bits to decode a 1-bit message using majority voting. I see that both stress time and increased copies reduce the error rate. The immediate repercussion of using multiple copies is the reduction in capacity. Increasing the number of copies provides diminishing returns on error reduction at the cost of reduced capacity.

Once the error rate is low enough, more efficient error correction codes are available. I illustrate this idea with experimental results where a Hamming code is added to the repetition code. Grouping bits into bins to decide the correct codeword is, in fact, a version of Hamming code. For instance, a 3-bit codeword of repeated 1-bit information is essentially a *Hamming(3,1)* with 000 and 111 valid codewords. In such error correction code, any codeword with Hamming weight 1 or 0 gets corrected into logic 0, otherwise, it is a logic 1. To show how a repetition and Hamming code work together, I add a *Hamming(7,4)* code on top of up to 17 copies of the payload. At the decoding phase, I calculate errors in each copy of the retrieved *payload* without any error correction, which gives us 6.5% mean error with 0.68% standard deviation. I apply this error in Equation 4.1 to calculate theoretical errors when I use multiple bits to decode the majority payload (the blue line in Figure 4.10). I observe that the *repetition code* closely follows theoretical predictions. Only *repetition code* itself brings the error to an absolute zero with 13 copies. The dotted black line

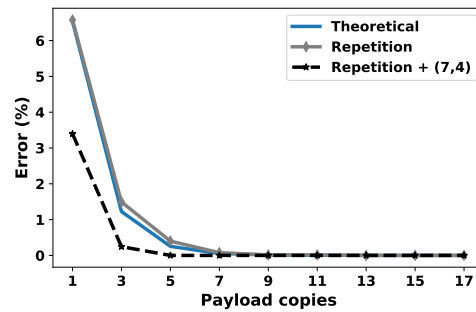


Figure 4.10: Application of Hamming(7,4) and repetition codes show how I can improve the error performance of a message hiding scheme that uses *Invisible Bits*.

in Figure 4.10 shows a Hamming(7,4) code combined with repetition code works. The combined codes work more efficiently because the combination drastically reduces the error within fewer copies, resulting in increased capacity.⁸

4.5.3 Performance Comparison

In this section, I show how *Invisible Bits* performs compared to Flash-based message hiding schemes. The noisiness of the Flash's analog characteristics is the source of plausible deniability. Occupying the entire Flash to store confidential information reveals its existence. Flash-based information hiding schemes suffer from low capacity because they allow only a few hundred cells per page to store hidden information. Besides, programming all the pages is challenging because encoded information in one page interferes with the neighboring pages' analog characteristics. As a result, they cannot select every page to encode information.

Additionally, Flash-based methods require grouping bits into a large bin (128-bit or more) to achieve low error. Considering these limiting factors, a Flash-based hiding scheme achieves 0.05% capacity [177]. When considering that on-chip Flash memories are usually reserved for program data, capacity is reduced further. For example, consider an MSP432P401 microcontroller

⁸Experiments suggest that the order of ECCs (repetition and hamming(7,4)) does not significantly affect the overall error rate.

Method	Ubiquity	Capacity	Resilience	Read stable
Zuck <i>et al.</i> [193]	●	●	●	●
Wang <i>et al.</i> [177]	●	●	●	●
<i>Invisible Bits</i>	●	●	●	●

Table 4.3: Comparison among on-chip information hiding techniques.

that has 256KB of Flash. Assuming that the entire Flash is available, write-time-based Flash hiding approaches can only transmit 131 bytes of information [177]. The more recent voltage-based technique [193] doubles this capacity by hiding information within the public data (*i.e.*, overlapping), but this assumes that the entire Flash is filled with data.

On the contrary, since encoding the payload in SRAM’s analog domain does not interfere with its digital operation, *Invisible Bits* has access to the entire SRAM—no matter how software uses it. Combined with its low error rate, *Invisible Bits* more than makes up for Flash’s 4:1 size advantage over SRAM. I provide a qualitative comparison between the on-chip steganography techniques in Table 4.3.⁹ I consider an encoded device with 6.5% (§4.5.1) and 5-copies *repetition code* to bring down the error at the same level ($<0.3\%$) across all approaches. Using five copies allows *Invisible Bits* to hide 12.8KB of payload ($20\% \times 64\text{KB}$), which is 100x of the Flash write-time-based method. Table 4.4 summarizes the errors when a hidden message contains a single copy and the settings of acceleration parameters for four devices.

Another advantage of *Invisible Bits* is that devices can be encoded in parallel. Given the importance of capacity in a steganographic covert channel, one can encode many devices and select the one with the least error. From my experimental results in Figure 4.6, a device with 2.7% error is possible. Such a device produces 33% information capacity with only three copies at less than 0.01% error rate. In such case, *Invisible Bits* hides 160x more information than Flash-based on-chip approaches.

⁹ ● = Excellent, ● = Very good, ○ = Good, ○ = Fair, and ● = Poor.

Device	SRAM usage	$V_{acc.}$	$T_{acc.}$	Bit rate	Encoding time
ATSAML11E16A	Main memory	4.8V	85°C	97.2%	16 hours
MSP432P401	Main memory	3.3V	85°C	93.5%	10 hours
LPC55S69JBD100	Main memory	5.5V	85°C	88.5%	24 hours
BCM2837	Cache	2.2V	85°C	79.2%	120 hours

Table 4.4: Summary of the results when I encode a message in different devices.

4.6 Invisible Bits in Action

Invisible Bits is a high-capacity on-chip covert channel that hides messages in the analog domain of SRAM. An immediate advantage of decoupling the message from regular computing space is digital plausible deniability. In this section, I expand my threat scenario and consider that an adversary inspects SRAM’s power-on state as a medium for information hiding. I show that using a stream cipher eliminates such threat and provides robust defense against analog-domain plausible deniability compromise. Then, I show an example of a covert and resilient end-to-end message hiding scheme using *Invisible Bits*.

Hiding information affects the symmetric distribution of an SRAM power-on state (Figure 4.3a), revealing the existence of a hidden message. I hide a message in an SRAM with error correction using both Hamming(7,4) and *repetition code*. In Figure 4.11a, I plot the post-encoded Hamming weight distributions of the power-on states of the devices. The distributions of hamming weights reveal the non-ideal distribution of SRAM power-on state, therefore, violates the analog-domain plausible deniability.

I attain analog-domain plausible deniability by using a stream cipher because it destroys the message’s structural properties, while being error neutral (*i.e.*, 1-bit error in the plain-text affects the 1-bit error in the ciphertext). I use AES-CTR to encrypt messages before encoding into an SRAM because standard AES encrypts a message and distributes the message bit uniformly. The counter mode allows decryption even if the ciphertext is partially corrupted due to error.

While the design remains the same as discussed in Section 4.4, I illustrate the application of *Invisible*

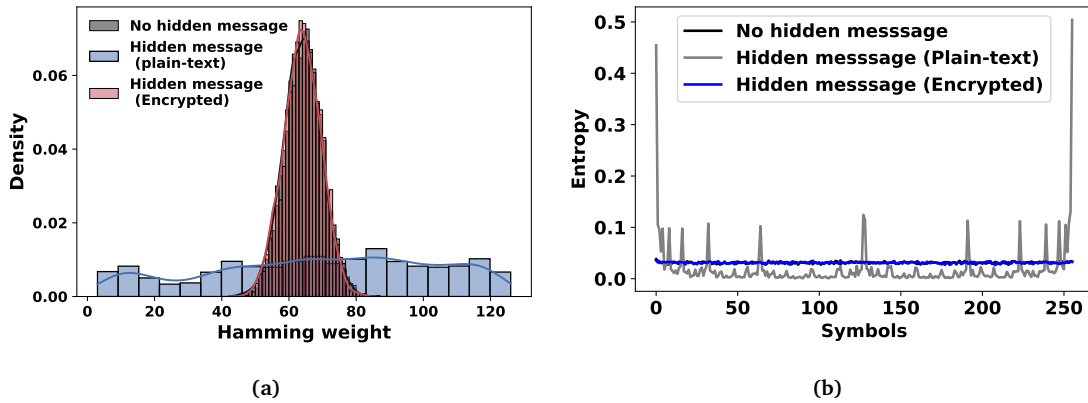


Figure 4.11: (a) Hiding information as plain-text, and ciphertext. (b) Shannon's entropy of SRAMs' power-on state. The normalized (by the number of symbols) entropy of an SRAM's power-on state is 0.0312, and the entropy becomes 0.0195 and 0.0312 in SRAMs that hide messages as plain-text and encrypted text, respectively.

Bits in steganography using a specific system shown in Figure 4.12. First, I apply a Hamming(7,4) on a message d and replicate the message and parity seven times to make it a hamming(7,1) code ($x = ECC(d)$). Second, x contains the data and parity bits, and AES converts these bits into a *payload* ($y = encrypt(x)$). Third, I store the *payload* into the SRAM of an MSP432P401 microcontroller. Fourth, SRAM undergoes the encoding process for 10 hours (§4.5), which changes the analog properties of transistors and, hence, influences the SRAM's power-on state.

The *Payload* extraction process follows the steps described in Section 4.4. The process begins with extracting the encoded SRAM's power-on states and using a shared key to decrypt the payload ($x' = decrypt(y')$). Finally, ECC retrieves the message (d'), which completes the steganographic message transfer.

I analyze the post-encode power-on state's properties to show that it is indistinguishable from an SRAM with no hidden message.

First, I run a hypothesis test on the post-encoded power-on distributions. I apply Welch's t-test on the sample of two classes of devices, with the null hypothesis being the chips have no hidden messages (*i.e.*, identical mean Hamming weight). I use a p -value heuristic to determine the significance of the t-statistic. My test's p -value is 0.071 (one-tailed); therefore, I cannot reject the null

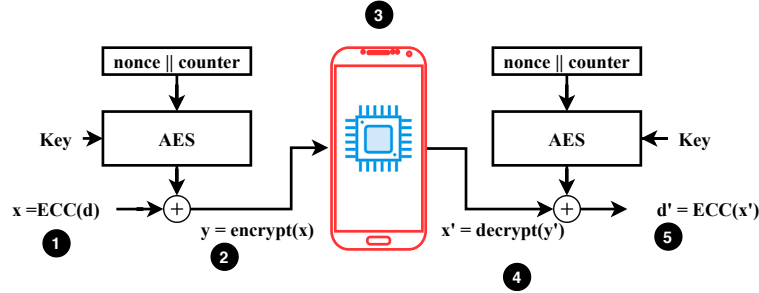


Figure 4.12: An end-to-end steganography system using *Invisible Bits*.

hypothesis. From this, I conclude that an adversary cannot statistically differentiate a chip with a hidden message from a chip with no hidden message (See Figure 4.11a).

Second, I study spatial autocorrelation to investigate whether an encoded device shows any difference compared to a device without any hidden message. In Table 4.5, I list spatial autocorrelation, the power-on biases for three classes of chips. I observe that chips without any encryption show spatial non-random patterns. On the other hand, the chips with encrypted hidden messages have similar *Moran's I* statistic and mean power-on bias.

Condition	Spatial autocorrelation	Mean power-on bias
Hidden message (no encryption)	0.502	0.537
Hidden message (no encryption)	0.409	0.535
No hidden message	0.001	0.500
No hidden message	0.004	0.501
No hidden message	0.002	0.501
No hidden message	0.007	0.502
No hidden message	0.009	0.500
Hidden message (encrypted)	0.001	0.500
Hidden message (encrypted)	0.008	0.501
Hidden message (encrypted)	0.002	0.499
Hidden message (encrypted)	0.001	0.499

Table 4.5: Spatial autocorrelation and mean power-on bias of SRAM embedded within an MSP432. Layering encryption on top of *Volt boot* makes devices with no hidden message indistinguishable from devices with messages encoded. Without encryption, devices with messages encoded have biases in their power-on state (e.g., more 1s than 0s) and spatially-correlated values. p -values are $\ll 0.05$ for all the measurements.

Third, I apply Shannon's entropy, a widely used metric to quantify uncertainty in information [82], in both pre-and post-encoded SRAM's power-on states. The entropy of a bit varies between 0 and 1; 0 being no uncertainty and 1 being fully random (e.g., fair coin flip). I divide the power-on state of an SRAM into byte granularity (symbol) and count the frequency of each 2^8 symbol (similar to

earlier work [63]). Analyzing 64K (size of the SRAM is 64KB) symbols from an SRAM's power-on state, I form a distribution and calculate Shannon's entropy using $\sum_{i=0}^n -P(x_i) \times \log_2(P(x_i))$. Here, $P(x_i)$ is the probability of i^{th} possible value out of n symbols. I plot the entropy distribution in Figure 4.11b calculated for SRAMs with no hidden message, hidden message as plain-text, and encrypted hidden message. The entropy distribution of an SRAM that hides a message as plain text deviates from the regular SRAM's power-on state entropy, whereas an encrypted hidden message closely resembles it.

Upon analyzing different statistical properties, I conclude that chips that encode encrypted messages are indistinguishable from 'clean' chips. Once the process is completed, the device containing SRAM becomes a covert and resilient information carrier with an ability to withstand both analog and digital domain steganalysis.

4.7 Discussion

4.7.1 Multiple-Snapshot Adversary

I assess a potential plausible deniability compromise by considering a stricter threat model where an adversary takes multiple power-on measurements (*i.e.*, snapshots) at different times to determine temporal discrepancies in the device's power-on state.

My recovery experiments show that some cells return to their original power-on values, and this recovery trend is greatest in the early post-encoded period (see Figure 4.7). I study the effect of capturing power-on states at different times in the recovery phase with the greatest power-on state change to assess whether it reveals the existence of an encoded message. The analysis starts with capturing the power-on states of a pre-encoded and post-encoded device. Then, I compare these measurements to the power-on states captured after one hour, one day, and one week of encoding. I calculate the spatial autocorrelation <0.01 with p -value $<< 0.05$ for all of the measurements.

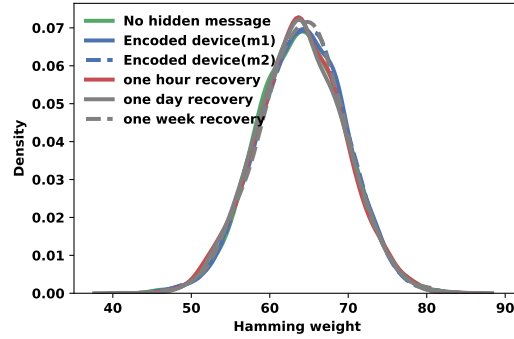


Figure 4.13: Distribution of Hamming weights captured at different phases and times of a covert communication.

This suggests that the spatial pattern of errors remains indistinguishable from an unencoded device during encoding and during recovery. The second statistic that would potentially reveal an encoded message is the distribution of Hamming weights for the SRAM when adjacent cells are grouped into fixed-size blocks. Figure 4.13 shows the distribution of Hamming weights for the measurements, where $m1$ and $m2$ indicate two back-to-back measurements of the same device. I expect some difference, even from subsequent measurements, as the cells with similar inverter turn-on voltages in an SRAM introduce noise in the measurement of power-on states (chapter 2). Following a similar analysis discussed in Section 4.6, I conclude that the difference in the snapshots captured at multiple points in time is indistinguishable from measurement errors. Thus, a multiple-snapshot adversary gains no special advantage in *Invisible Bits*.

4.7.2 Aging Complex Systems

High voltage plays a major role in accelerated aging (§4.2), and applying such voltage in a complex device requires special consideration on its power supply system. When a device is simpler such as MSP430, I directly apply voltage to its V_{dd} line. Complex devices (e.g., Raspberry Pi), however, regulate voltage before feeding it to the core (and SRAM). These circuits prevent elevating the supply voltage directly from the V_{dd} line. My observation is that most devices use switching regulators at

run-time to save energy because these regulators are inherently efficient in switching voltage levels. By design, these regulators need large passive components such as inductors. Since these passive components have a large footprint, the most practical design decision is to place them in the PCB and use a pin to connect them with the internal supply line. I exploit this pin (similar to *Invisible Bits*) to reach the core supply line directly and elevate the core voltage.

4.7.3 Device Selection & Capacity Considerations

The required capacity and the least tolerable error in the reconstructed message guide the device selection for *Invisible Bits*. While these two requirements are somewhat inversely proportional, the *baseline error rate* is determined largely by the SRAM's technology node. The manufacturing technology node determines the maximum allowable stress level, which eventually determines the minimum noise in the hidden message. SRAM manufactured at smaller technology nodes are more susceptible to aging as the electric field density in these devices is higher as compared to larger devices [78]. Thus, newer devices that can endure higher voltage and temperature for longer time provide the lowest error rate, which makes the most of the available SRAM cells. That being said, desired capacity is the driver behind device selection.

Finding average capacity in each device class requires testing a large sample of devices with full system implementation, including appropriate ECC selection for a message. I provide a theoretical analysis to illustrate the capacity and error trade-off using Table 4.4 and augmenting it with ECC (*i.e.*, repetition codes and Hamming (7,4)) to calculate errors at different capacity-levels. Figure 4.14 shows how the error rate affects the capacity across device classes.

Regardless of their locations, Flash memories tend to be larger memory devices compared to SRAMs. When I compare on-chip SRAM with off-chip Flash memories, the size difference is orders of magnitude larger. While the density of hidden information remains higher for SRAM-based steganography, Flash-based methods hide more information when off-chip memories are considered because inherently Flash-based storage media, such as SSDs, are much larger than on-chip SRAMs. Being

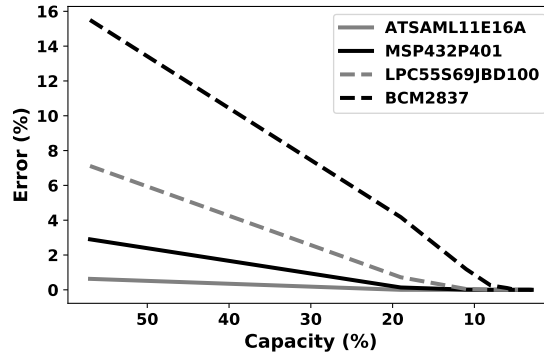


Figure 4.14: Error and capacity trade-off. I encode a single copy message in each device and simulate Bernoulli trials for different payload copies and Hamming(7,4) error correction code.

external enables Flash to have high capacity, but it also makes it trivially separable from any system that it is attached to.

4.7.4 Adversarial Aging to Inject Noise

In this section, I discuss a scenario where an adversary attempts to erase or inject noise in the hidden message in a device's analog domain by exposing the device to aging. It is possible to inject noise in the hidden message as aging-induced degradation has both reversible (NBTI) and irreversible (HCI) components. While it portrays an unlikely scenario where authorities (*e.g.*, border security) expose a large sample of devices to invasive aging for a prohibitively long time, I explore this possibility with an experiment to understand the impact of such adversarial aging. In this experiment, I expose an encoded MSP432 to aging with a power-on state for 1 hour. This increases the message reconstruction error 1.12 $\times$. When aged again for 1.5 hours with the power-on state, the error comes to its pre-adversarial aging state (error 0.98 $\times$). Cells with symmetric inverters are more susceptible to aging compared to asymmetric cells, and aging for short times injects noise by changing the power-on state of the symmetric cells. The receiving party can reduce the impact of noise by aging it in a similar way.

This scenario above considers an adversary who overcomes the following barriers to inject noise on

a device that *may* contain a hidden message in SRAM. First, aging a device involves physical tampering: an adversary needs to open up a device, decouple its power supply system, and externally elevate the voltage of the core (§4.7.2). Second, since *Invisible Bits* encodes information in the analog domain by exploiting *physical* changes in transistors, reversing the aging-induced degradation is *prohibitively time-intensive*. Third, the next challenge is to improve the effectiveness of adversarial aging, *i.e.*, maximizing the injected noise. As discussed in Section 4.2, aging pushes the power-on state of a cell to the complement of the stored logic state, and therefore aging an encoded SRAM with a power-on state injects the maximum number of noise bits. Setting a device in this state (*i.e.*, SRAM with its power-on state) *requires firmware tampering* because as soon as a device boots, the system software clobbers the power-on state of its SRAMs.

4.8 Related work

4.8.1 Hardware-based steganography

The closest related work to *Invisible Bits* is *Flash program time-based* message hiding method [177]. This method deliberately stresses a group of cells to encode information in them. The program time of cells is distributed over a long-tailed spectrum, and so changing a few cells' program time becomes indistinguishable from non-encoded cells. A group of 128-bit cells encodes 1-bit information, and addresses of the cells that are grouped are encrypted using a symmetric key cipher. Zuck *et al.* [193] uses the voltage level of a flash memory cell as the medium for hiding information. The technique uses two passes to encode a logic state. The first pass stores encrypted cover data, and the second pass selects a few cells from the same public bits, preferably logic 1, to encode hidden data. To encode 0, a non-programmed bit, cells currently holding public data are incrementally charged beyond their preset voltage level. As long as the cover data is not erased or re-programmed, the hidden data remains stored, meaning—unlike *Invisible Bits*—the encoded data is fragile. An active adversary can promptly stop covert communication by copying the encrypted cover data and re-

programming it again without modification. In such a case, data is lost. Another limitation of this approach is that it is not plausibly deniable at the digital level as the encrypted cover data signals that the device carries some secret message, even if the actual message is in the analog domain. On top of these limitations, both methods suffer from low capacity when compared to the *Invisible Bits*, because they have to deal with relatively large operational noise margins, where the tightly packed nature of SRAM cells filters operational noise as common-mode noise.

Srinivasan *et al.* exploit Flash transport layer (FTL) and over-provisioning of solid-state drives to hide information in the physical layer of a mass storage [162]. DEFY introduces multiple security levels in the data handling of FTL [127]. The existence of higher security level data is not visible to the lower security level. Therefore, a victim can release the lower security level data to deny higher security level data. These methods suffer a general problem, unintentional overwriting, that stems from the coexistence of public and private information [27, 158, 159]. Even worse, Jia *et al.* [74] shows that adversaries can detect hidden messages in the lower-level security and copy the data and rewrite it back.

4.8.2 System-Level Steganography

A large body of work exists where researchers exploit unnoticeable changes in numerous media, such as digital objects, network protocols, file systems, and cyber-physical systems (CPS). The most popular system-level steganographic technique is embedding a message within an image's pixels, such that it is imperceptible by an adversarial steganalysis [42, 152, 178]. A common method is to encode the message in the least significant bit of a subset of pixels; the larger the subset, the less plausibly deniable the encoding becomes, but the higher the capacity. Audio and video signals add a covert temporal channel to still images. Thus, messages can be encoded in their time, frequency, and wavelet domains [57, 175, 185]. Another medium with temporal aspects are network signals; one can exploit network protocols for covert communication by using them beyond their intended design, such as injecting messages into the empty places in the protocol headers [5, 146, 187].

These system-level steganographic mediums tend to be very low capacity; otherwise, they lose their plausible deniability.

4.8.3 Infrastructure-Level Steganography

Wendzel *et al.* exploited CPS, such as building automation system (BAS) to exchange and store hidden information [180, 181]. The authors show that there are at least two possible ways, unused registers and actuators, state one can store information in a BAS. The parties involved in information exchanges in such a way must be able to access BAS, which may not be possible for most practical applications.

4.9 Conclusion

This chapter uncovers an on-chip steganographic medium based on SRAM’s analog-domain properties, namely the relative threshold voltage of the cross-coupled inverters that compose every SRAM cell. To exploit this new covert channel, I design and implement *Invisible Bits*, a steganographic system that uses accelerated, directed transistor aging to encode information into SRAM’s analog-domain properties and reveals these changes at the digital level by capturing SRAM’s power-on state. My evaluation, using modern computing devices, shows that *Invisible Bits* increases capacity to over 90%, a 100x improvement over existing on-chip steganographic techniques—while retaining both digital- and analog-domain plausible deniability. In addition to improved capacity, my evaluation also shows that *Invisible Bits* improves resilience as it maintains the vast majority of the encoded data after being shelved for a month and even after being used continuously for over a week. Lastly, the results of my evaluation to guide future users on the most appropriate error-correcting codes and ciphers to layer on top of *Invisible Bits*.

Chapter 5

Exploiting Long-term Data Remanence to Exfiltrate Secrets from ARM TrustZone Guarded SRAM

5.1 Introduction

In 1988 Mark Weiser coined the term ubiquitous computing to describe his vision of a future society where computers are deeply-ingrained, ever-present, yet invisible [179]. We are rapidly fulfilling this vision as transistor scaling brings us into the Internet of Things (IoT) era of computing and brings smart dust [110] within reach. As a consequence, we are already seeing critical infrastructure [99], medical devices [148], and defense systems [24] become more dependent on IoT devices. This dependence will only increase in the future as we fulfill Mark Weiser’s vision.

Given society’s increasing dependence on IoT-scale systems, their security is paramount. Security concerns for this class of devices extend beyond more traditional devices to emphasize the physical domain. This is due to their ingrained nature, where physical access is the common case. Thus, one of the grand challenges of security in the era of ubiquitous computing is protecting secrets when the attacker can hold a device in their hands.

An accepted solution to the threat of physical access is to shrink a system’s security perimeter to the chip that performs the security-critical computation. This prevents an attacker from interposing

and capturing secrets while they are in-flight between system components or at-rest in a less-secure component. For example, the cold boot attack captures secrets while they are at-rest in DRAM chips [59]. In the cold boot attack, the attacker leverages the data remanence effect of DRAM (enhanced by reducing the chip's temperature) to transfer the DRAM chip(s) from the victim's system to the attacker's system—without corrupting the data. Once the attacker installs the DRAM into their system, they employ forensics tools to hunt for secrets (e.g., cryptographic keys). To eliminate such attacks, researchers advocate fully on-chip secure computation, where plaintext secrets are never allowed off-chip [32, 40, 126].

Recently, researchers and processor vendors alike further increase system security by constricting the security perimeter to hardware-enforced partitions within the chip (called enclaves), creating a Trusted Execution Environment (TEE). TEEs increase security by protecting against bugs in software executing outside the TEE—even if it is privileged software. The most widely deployed TEE is ARM TrustZone [10]; most modern ARM processors offer the TrustZone extension, including both microcontroller- and application-class devices. TrustZone provides hardware support to software by switching a core between a *Secure World* and a *Normal World*, where the *Secure World* can access the entire system, but the *Normal World* cannot access *Secure World* resources. At run-time, TrustZone hardware checks the access permission of each transaction to ensure that it is within the appropriate security boundary; attempts to access *Secure World* resources from the *Normal World* trigger a *HardFault*. Fully on-chip crypto-operations combined with TEE support, as exemplified in as *Cache assisted Secure Execution (CaSE)* [190], create the most comprehensively secure software execution environment today.

Undeterred, researchers continue to identify weaknesses that processor vendors must address in their defenses. Attacks on TrustZone focus on side-channels created by implicitly shared physical resources. The first attack type exploits shared cache and TrustZone's security-state-oblivious eviction policy [29, 95]. The second attack type exploits the energy management policy, which applies across both *Secure* and *Normal* worlds, to induce faults in security-critical software executing in the *Secure World* [130, 164]. The solution to these attacks is a further reduction in the security perime-

ter, but an open question remains, "*Is security perimeter reduction sufficient to eventually secure the system?*"

I show that the current focus on placing temporally-adjacent restrictions is insufficient by uncovering and exploiting the threat of long-term data remanence. Data remanence is when a memory device retains information past when it is assumed to no longer exist. Short-term data remanence (due to capacitance in the circuit) is well-studied, having led to cold-boot-style attacks on both DRAM [18, 59, 167] and SRAM [160, 161, 169, 188]. In a cold-boot-style attack, attackers leak secrets by resetting software or moving memory between machines, before the memory loses state. The attackers can then use their own software to copy the latent data in the memory device—without restrictions of the original software/system. More recently, short-term data remanence attacks have been used to clone SRAM-based Physical Unclonable Functions (PUFs) [188] using the same process. **Given the threat of data remanence, TrustZone eliminates such attacks by wiping all secure memory as part of all *Secure World* software updates.**

I show that wiping memory is insufficient as circuits “remember” data for much longer through subtle changes to their analog-domain properties—changes that depend on how the software uses the circuits. Previous research reveals that circuits undergo a process called aging [2, 80, 90], and this aging is data-dependent [58]. Researchers have leveraged data-dependent aging in memory circuits and processing cores as a denial-of-service attack on SRAM-based PUFs [139] and processors [76] and to reveal a small fraction of RSA key bits [8, 67, 112] in simple discrete SRAM chips. This paper builds on those papers, showing how device aging can be controlled and accelerated exponentially to reveal up to 98% of secret data stored in on-chip SRAM—even in the most recently produced ARM TrustZone devices. My results motivate extending existing short-term data remanence defenses to include long-term data remanence.

There are two challenges that I address to create a practical attack out of long-term data remanence:

- ❶ measuring analog-domain changes in an efficient and non-destructive manner, i.e., a digital measurement, and
- ❷ increasing the rate of analog-domain change such that secrets measurably

change the system at the digital level. I solve the first challenge by leveraging a unique property of the memory used to hold secrets on-chip SRAM: When SRAM cells power on at boot time, its construction leads to a hardware-level race condition to set the initial value of the cell (i.e., its power-on state). The relative analog-domain properties of the logic gates at the heart of the SRAM cell influence this hardware-level race. Thus, by capturing the power-on state of an SRAM cell, I get a digital window into its analog-domain properties. As I show, these analog-domain properties depend on the value held by the SRAM cell—which software dictates—in a process known as transistor aging; meaning that software gradually imprints itself and its data into SRAM’s power-on state over a device’s lifetime. Since it is not practical to wait for secrets to imprint themselves over several decades, I solve the second challenge by using non-invasive stressors (i.e., temperature and voltage) to accelerate the imprinting process over 100,000,000x. **The resulting foundational capability is the power to pause software, imprint software’s secrets into SRAM for less than 24 hours, and reveal the imprinted secrets through power-on state samples with up to 98% accuracy.**

This chapter makes following technical contributions:

- I demonstrate that modern SRAMs are sensitive to accelerated stress, which allows an attacker to retrieve on-chip secrets without sophisticated tools (§5.5).
- I demonstrate how SRAM’s aging creates a low-cost side-channel for on-chip cryptographic key exfiltration (with 97.2% accuracy) (§5.6).
- I show how an attacker clones proprietary firmware (with up to 95.82% accuracy) that is protected using TrustZone-enforced on-chip execution (§5.7).
- I extend the attack to general-purpose processors and show how cache memories are susceptible to long-term data remanence attacks (with 79%- 93% accuracy)(§5.8). I combine all three attack scenarios in a commodity Cortex-A72 processor to demonstrate the generality of the proposed attack (§5.9).

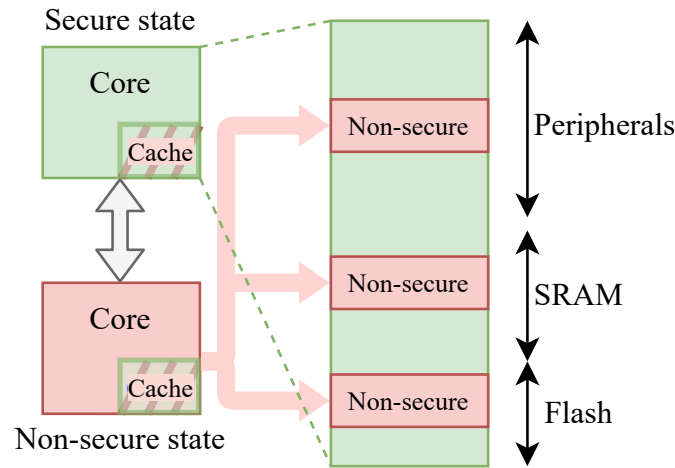


Figure 5.1: An abstract view of a TrustZone-capable processor and its different access permissions throughout the system. In an ARMv8-A architecture, caches are shared between *Secure World* and *Normal World*. ARMv8-M architecture allows configurable partitions in the processor’s memory map.

- I discuss the defensive landscape from both software and hardware levels (§5.10).

5.2 Background

TrustZone, a widely deployed security extension for ARM-based devices, facilitates TEE for secure software by applying hardware-enforced barriers among applications; my attack reveals on-chip secrets from such systems exploiting transistor aging and SRAM’s power-on state. Aging-induced change in the power-on state is slow and takes decades to leave exploitable data traces under normal operation. From an attacker’s perspective, artificial stress eliminates these downsides of the natural aging response of a device. This section provides background on TrustZone, SRAM’s power-on state, and aging acceleration methods to imprint software secrets into an SRAM’s analog domain.

5.2.1 ARM TrustZone

A TrustZone divides a system into the *Secure World* and *Normal World* where the same physical core executes secure and non-secure applications in a time-sliced manner by switching between the *worlds* [10]. The secure state has full system access, while the non-secure state (*i.e.*, *Normal World*) only has access to non-secure memory regions. An NS flag identifies the security state of the processor, and the *AMBA* bus carries this flag to all the system components to maintain hardware-enforced isolation between secure and non-secure transactions. The tag of each cache line carries an NS security flag to indicate whether the line belongs to a *Secure World* or a *Normal World*. It is architecturally impossible to access a secure cache line with non-secure access [13]. While in a *Secure World*, a core can switch to *Normal World* and vice versa by executing specific instructions such as SG and SMC for Cortex-M devices and Cortex-A devices, respectively. Figure 5.1 illustrates an abstract view of a processor with TrustZone extension.

Cortex-M devices allow *Normal World* applications to request services from *Secure World* using Non-Secure Callable (NSC) functions. The NSC functions are ‘black box’ to a non-secure application, and any data inside these functions are not exposed to the *Normal World*. A call to these functions switches the core into *Secure World*, and then it executes the function.

A Cortex-M device can be configured into multiple levels of security settings in its lifetime. The hardware isolation paves the way to allow independent software development for *Secure World* and *Normal World*, providing security to the proprietary software and its data. For example, one user can design only secure software and program a device before handing it over to another user. The second user programs the non-secure part of the system without any access to the secure area other than through NSC functions. The only way to gain upper-level security access is by erasing the entire chip or having an authorization key from the secure user. *UntrustZone presents an innovative approach to accessing secrets across security boundaries by leveraging SRAM’s analog behavior* (§5.6, §5.7, and §5.8).

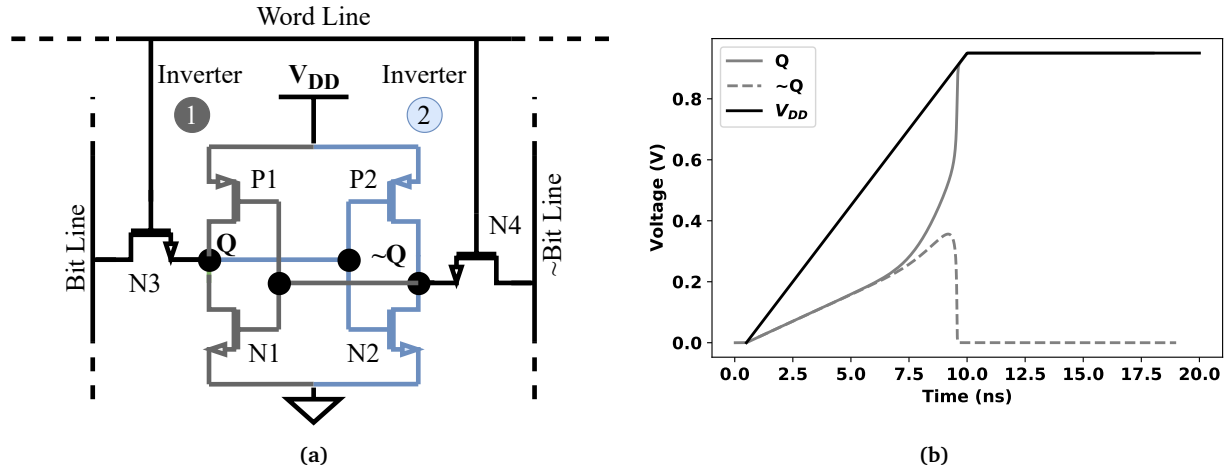


Figure 5.2: (a) Schematic of a conventional 6-transistor SRAM cell and (b) power-on behavior of an SRAM cell at the 22nm technology node.

5.2.2 SRAM Power-on State

Figure 5.2a illustrates a typical 6-transistor schematic of an SRAM cell. The cross-coupled CMOS inverters store a bit of data through positive feedback. Outputs of Inverter ① and Inverter ②, Q and ~Q, respectively, are connected to each other's inputs, forming a loop. When *Word Line* is asserted, transistors N3 and N4 provide access to the data stored in the loop. These two transistors remain off unless the address decoder performs a reading or writing operation.

At power-on, nodes Q and ~Q are the ground state and gradually settle their respective steady-state voltages; hardware race condition between Inverter ① and Inverter ② determines the final values [38]. Since one inverter feeds the input of the other inverter and outputs are the same voltage at power-on, the hardware race condition puts these two nodes into opposite logic states. We simulate this behavior using *HSpice* (@22nm predictive technology model [15]) and plot the resulting waveform in Figure 5.2b. The output nodes, Q and ~Q, follow the supply voltage ramp and resolve into opposite logic states long before the V_{DD} ramp settles.

Inverter ① and Inverter ② are electrically identical as they are designed to be mirrored versions. A cell with such symmetric characteristics produces random power-on values depending on the operating condition of the cell. Most cells, however, power on into a defined logic state due to

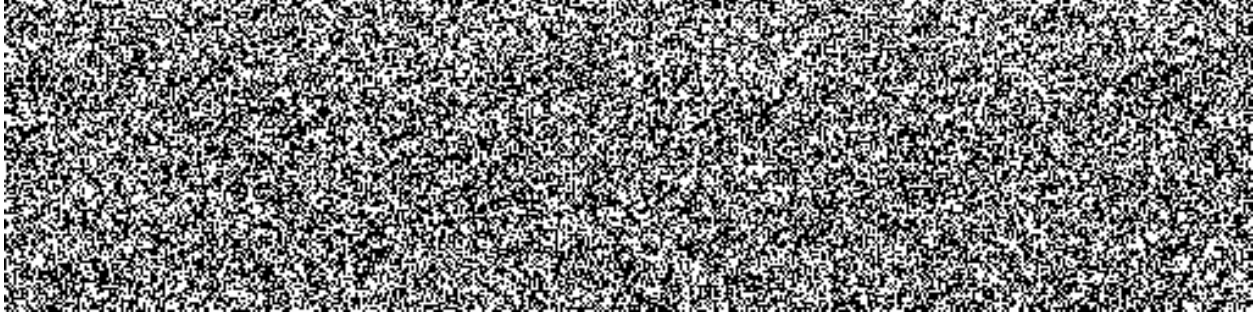


Figure 5.3: Heatmap of a SAML11 device’s power-on SRAM state.

the asymmetry introduced by manufacturing-time process-related variabilities [3, 189]. As a result, some SRAM cells show random power-on values across trials (*i.e.*, *weak cells*), while others show a strong bias towards one logic state over the other (*i.e.*, *strong cells*). A cell’s bias 0.5 indicates that 50% of the time, it powers on as logic 0, and in the other 50% of the trials, it becomes 1. We dump the power-on values of a SAML11 device [114] multiple times and construct a distribution of the bias of each cell. Approximately 10% cells show weak characteristics ($0 < bias < 1$), and 90% cells strongly prefer either logic 1 or 0. Figure 5.3 illustrates a 128×512 -bit snippet of the SAML11’s SRAM power-on state (majority-voted). In the next section (§5.2.3), *I will show how an attacker can change this hardware race condition (and power-on state) to reveal the value stored in an SRAM cell.*

5.2.3 Analog-domain Changes and Power-on State

As transistors operate, they go through electrical and thermal stress, which affects their performance. Physical phenomena such as Hot Carrier Injection (HCI) [21] and Bias Temperature Instability (BTI) [2, 90] are responsible for a transistor’s performance degradation over time, which is known as *aging*. Aging results in a gradual increase of the threshold voltage ($|V_{th}|$) and reduced carrier mobility (μ) of transistors [150]. Hot carrier injection is responsible for aging when a transistor switches between logic states, and switching frequency determines the magnitude of HCI-induced aging [150]. Bias temperature instability ages a transistor as long as it remains ON (*i.e.*, gate to

source voltages $|V_{gs}| > |V_{th}|$). In an SRAM cell, regardless of the stored logic state, two transistors experience BTI-induced aging: in Figure 5.2a, $Q = 1$ ages P1 and N2 while $Q = 0$ ages P2 and N1. A MOSFET turns ON in the *saturation region* where the drain current is $I_d \propto (|V_{gs}| - |V_{th}|)^2$ [147]. A transistor's drain current I_d reduces over time because the aging-induced degradation increases the threshold voltage ($|V_{th}|$), slowing it down. Note the impact of BTI in PMOS (NBTI) is much more dominant than BTI's effect on NMOS (PBTI) beyond 45nm Technology [78]. However, PBTI has a significant impact on aging a modern transistor with high-K dielectric and metal gate [83].

Aging affects the switching speed of a transistor, which in turn influences the outcome of the hardware race condition at an SRAM's power on. That is, as the power-on state depends on this race condition of an SRAM's startup, *aging potentially impacts the state* at which the cell eventually resolves its transient state. For example, let us assume the cell in Figure 5.2a powers on at logic 1 because Inverter ① turns on faster and pulls the node Q to V_{DD} . If I leave the cell operational in this state (with $Q = 1$, and $\sim Q = 0$), P1 undergoes static stress because $|V_{gs}| > 0$ (I ignore the effect of NMOS aging for this example). According to the charge trapping/de-trapping theory of BTI [183], the threshold voltage of P1 ($|V_{th_{P1}}|$) logarithmically increases over time, making P1 slower. When this analog-domain change makes Inverter ② faster than Inverter ① (*i.e.*, $|V_{th_{P1}}| > |V_{th_{P2}}|$), the power-on state of this cell flips (*i.e.*, the cell's post-stress power-on state is 0). On the other hand, if the cell undergoes aging with logic 0, the initial speed difference between P1 and P2 increases in the same direction; transistor P2 becomes slower, reinforcing the power-on state (logic 1) of this cell. The complement of the power-on state indicates the logic state it holds during the aging process. *Thus, given sufficient operation (aka stress), the power-on state of an SRAM cell reveals the software value held by that SRAM cell during operation/stress.*

5.3 Attack Overview

In the overarching threat model of the attack, I assume *an attacker with physical access* to a device that holds secrets on-chip guarded by countermeasures (e.g., TrustZone) to conventional hardware- and software-level attacks. We create target-information- and SoC-specific threat models to represent distinct attack scenarios, each fulfilling the overarching threat model.

We observe that SoCs allow access to uncontaminated SRAM power-on state and core voltage pins, and these architecture/circuit-level features are exploitable to induce aging in SRAM (and imprint secrets). This observation leads us to develop *UntrustZone* attack. Its secret retrieval accuracy depends on aging acceleration (1) voltage, (2) temperature, and (3) stress time (§5.5.2).

The attack begins with identifying the suitable voltage pin(s) and corresponding off-chip power delivery components, followed by measuring the nominal voltage at the pin(s). Once these attack parameters are available from experiments with guidance from the publicly released device datasheet, I place the target device in the stress condition (*i.e.*, accelerated aging). The secrets imprint into the analog domain of SRAM during the ‘burn-in’ effect (*i.e.*, aging) of the victim device, and after a full chip erasure or privilege change (due to TrustZone enforcement) SRAM’s power-on state reveals the secure-access-only information to non-secure state. To demonstrate the potential threat of long-term data remanence, I conduct the attack on three recent commodity devices and show that fully on-chip execution with support from hardware-enforced security isolation is insufficient against the proposed attack. The following is a summary of the attacks and their outcomes:

- **Exfiltrate cryptographic key from TrustZone:** In the first attack, I illustrate how *UntrustZone* exposes a cryptographic key from a system that runs fully on-chip computation, guarded by TrustZone. We use a popular microcontroller, SAML11, which is marketed on its security features [115]. The victim application uses on-chip crypto—in addition to hardware crypto accelerators—to protect the plaintext. Even with these protections in place, I capture the key with **97.2% accuracy** (§5.6).

- **Exfiltrate proprietary firmware from TrustZone:** In the second attack, I target proprietary firmware designed to be secure against off-chip cloning. Despite being fully on-chip while the CPU performs computations, *UntrustZone* is able to expose this firmware’s instructions and data from the SRAM with up to **95.82% accuracy** (§5.7). For this demonstration, I use a dual-core LPC55S69 microcontroller [124].
- **Exfiltrate secrets from cache:** In the third case, I extend the attack to target more complex SoCs (Cortex-A53 and Cortex-A72 from Broadcom), which use private caches as nonvolatile secret storage (§5.8 and §5.9). We steal secrets from caches of a Cortex-A53 and Cortex-A72 processor with 79% and 93% accuracy, respectively.

When exploring software mitigation, I observe that TrustZone does prevent access to secrets by erasing memory when security access level changes, but long-term data remanence still reveals secrets through the analog domain. Software-level defenses to *UntrustZone* include preventing access or erasing the power-on state during an SoC’s boot phase and/or scrambling SRAM data at runtime to reduce an adversarial *accelerated burn-in* effect. These defenses have shortcomings, such as eliminating security applications of SRAM power-on state and boot speed reduction. On the hardware side, system and SoC designers have options to limit electric and thermal acceleration and control debug access (once TrustZone has been deployed). While effective, these mitigation techniques pose many challenges, such as strictly modeling runtime core voltage fluctuation and enforcing debug authentication. To facilitate future defenses, I provide a detailed qualitative analysis of the trade-space between software- and hardware-based defenses in Section 5.10.

5.4 Identifying Attack Enablers

An attacker needs access to an SRAM’s uncontaminated power-on state and a way to accelerate the secret imprinting to launch a long-term-data remanence attack in a reasonable time frame because normal circuit aging takes decades to ‘burn-in’ data in SRAM’s analog domain [106]. I apply high

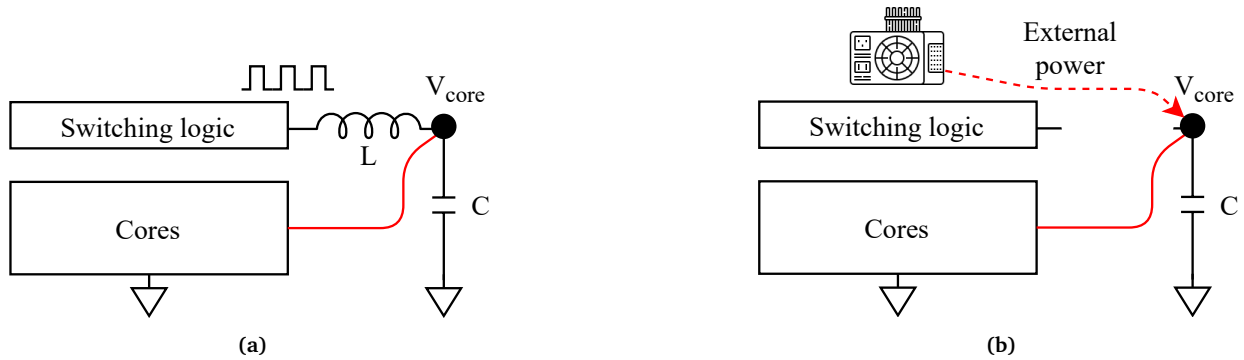


Figure 5.4: Illustrates the target system’s (a) nominal power supply configuration where a switching regulator supplies the SoC’s cores. (b) My attack configuration, where I remove the inductor (L) and connect an external voltage probe to control the victim’s voltage.

voltage and temperature to accelerate aging so that I can imprint data to and extract data from SRAM’s analog domain within hours. While prior work shows that higher than nominal voltage accelerates circuit aging, they fail to recognize the circuit-and system-level complexity associated with the modern SoC, which prevents elevating SRAM voltage [8, 67, 139]. *This leaves numerous questions unanswered, for example, how to elevate a cache’s voltage to attack cache-based on-chip computation while keeping the processor from resetting the data on the cache?* That is, attack execution techniques on a discrete SRAM chip and a complex SoC are significantly different; *UntrustZone bridges this gap* by exploiting the power delivery network of a modern SoC to provide access to the internal power rail and accelerating the secret imprinting process. The following discusses three hardware and software challenges that I address to execute *UntrustZone* on real-world devices.

C1: Overdrive SRAM’s power bus.

Electric and thermal stresses during regular operation are responsible for a transistor’s performance degradation (*i.e.*, aging), but from an attack perspective, natural aging is slow and impractical for exposing secrets [106]. Exposing a device to higher voltage and temperature induces stronger electric and thermal stress expediting aging.

Modern microprocessors are equipped with brown-out detectors and high-voltage protection circuitry to prevent under and over-voltage in their power buses. Applying an increased voltage di-

rectly at the V_{DD} pins does not increase the SRAMs voltage because of internal protection circuitry and voltage regulators. Such an attempt only overheats a device due to a significant voltage drop in the linear regulators (or voltage clipping circuitry). However, this does not impact the memory domain voltage in a way that would induce accelerated imprinting.

In SoCs, however, the internal power supply system divides logical blocks (and physical blocks) which are electrically reachable [104] from external pins bypassing protection and regulator circuitry. SOCs need dedicated external voltage pins (*i.e.*, V_{core} to supply power to the core components) to filter noise generated from fluctuating power demand or to supply power from efficient switching regulators instead of linear regulators.¹ The passive circuit components needed for these operations are large and must be placed on the Printed Circuit Board (PCB); I elaborate on this concept as follows:

Noise-free power delivery: The power consumption of a processor depends on micro-architectural events, and a power delivery system must handle the fluctuation generated by a variable workload. High current fluctuation generates voltage drop in the supply input (V_{DD}) due to parasitic inductance from the die, package, and PCB (*i.e.*, $V_{drop} = L \frac{dI}{dt}$) [84, 85].

Efficient power delivery: The most common reason for leaving internal voltages (*e.g.*, V_{core}) exposed comes from a system’s need for energy efficiency at run-time. A device can run from both linear (*e.g.*, LDOs) and switching regulators (*e.g.*, buck converters). Switching regulators are inherently more efficient for meeting variable demands as they control voltage levels by changing the *duty cycle* of a pulse which is then filtered out to a *stable voltage* using passive electronic components such as diodes, inductors, and capacitors, before feeding it to the CPUs and memories.

As mentioned, the passive components are placed outside the SoC to save the die area, manage heat, avoid timing violations, and maintain a wholly-digital process. Figure 5.4a illustrates a power supply seen in typical SoCs. The duty cycle of the generated pulse from the switching circuit controls V_{core} . Caches (and most other SRAM) are tightly coupled with the CPU core and, by default,

¹Microcontrollers usually boot from internal Low Drop-Out (LDO) regulators, then switch to switching regulators (if available) to save energy due to their power efficiency compared to LDOs.

draw energy from the same internal power rail (i.e., V_{core}). My observation is that removing the inductive component while keeping the supply voltage at a nominal level using an external power supply (Figure 5.4b) allows direct control to the SRAM power supply rail, bypassing complex power supply circuits that sit between V_{DD} and the internal power rail that feeds SRAM.

C2: Capture SRAM's power-on state from software.

Processors allow access to SRAM's power-on state like any other memory read operations if it is the main memory (e.g., microcontrollers) or memory-mapped cache (e.g., RISC-V architecture [155]). Some processors do not provide access to their caches directly using a memory-mapped interface, such as *Cortex-A profile* processors. However, I find that these processors provide coprocessor instructions to debug cache coherency issues and low-level memory translation errors [12]. ARM coprocessors are capable of performing maintenance operations without enabling the caches and regardless of the cache lines' validity. These operations allow direct access to the uninitialized states of cache-lines even if they are invalid at startup.

C3: Reduce contamination of SRAM power-on state.

The ability to read SRAM's state directly is not enough; an attacker must be able to capture SRAM's power-on state before it is overwritten—intentionally or otherwise. I observe that processors do not erase the power-on state as part of the hardware reset sequence, as it does not provide any clear benefit. In fact, there are a few downsides to executing power-on state erasure through hardware or software.

On-chip SRAMs can be large, ranging from a few kilobytes to a few megabytes, and distributed in many banks. Iterating over a large memory at a word or line granularity adds significant time to a processor's boot-up. Also, it is not clear (outside of preventing my attack) what value clearing all of a device's SRAM has.

Another deterrent for SRAM initialization is the fact that SRAM power-on states have numerous security applications. First, *weak cells* in an SRAM array capture randomness from the operating

noise, which a system uses as a source for random number generation (TRNGs) [63, 131]. Second, *Strong cells* carry unique hardware characteristics that serve as the foundation for Physical Unclonable Functions (PUF) [48, 97, 174], fingerprint generations [28, 63]. Third, the behavior of these cells changes over time due to aging, which is leveraged in counterfeit IC detection [100, 182]. Access to the uncontaminated SRAM power-on state is essential for these applications. These reasons are why processors do not initialize the SRAM [9] as part of their boot-up sequence.

In summary, unrestricted electrical access to an SRAM’s power bus and software-accessible, uncontaminated power-on state lead to my analog-domain attack that makes it possible to leak secrets from a secure memory area.

5.5 Attack Evaluation

This section discusses my baseline experimental setup, and the following sections (§5.6, §5.7, §5.8) show the execution of long-term data remanence attacks on three different scenarios.

5.5.1 Setup and Target Devices

Figure 5.5 illustrates a high-level block diagram of my experimental setup, which automates the following tasks:

- Controls experiment temperature and a target’s core voltage at different stages of the attack execution.
- Loads software to a target when needed.
- Provides access to the target’s memory through a debugger or in-system power-on state extraction software.

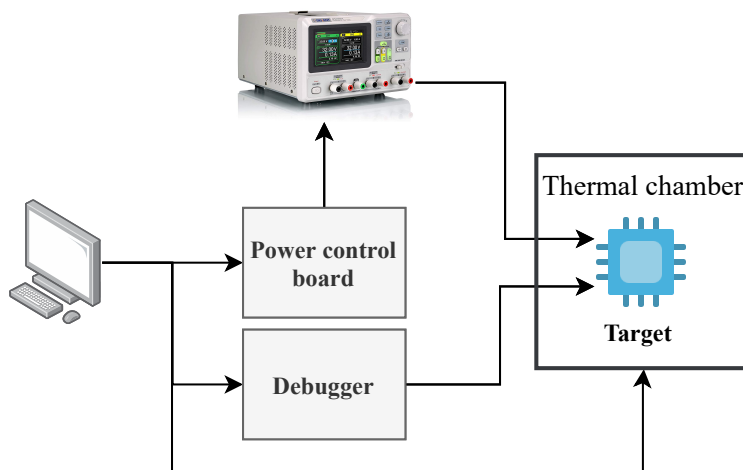


Figure 5.5: A diagram illustrating the high-level configuration of the experimental setup.

Figure 5.6 shows the essential parts of my experimental setup. The in-house power control board manages different voltage levels while the debugger provides access to the target’s internal resources during power-on state extraction. To induce controlled thermal stress, I use a thermal chamber from *TestEquity* [166]. As mentioned in Section 5.2, some cells are susceptible to operational noise (e.g., temperature, power supply noise, etc.) and behave unpredictably. I model their probabilistic behavior using multiple trials (i.e., power cycles) and calculate a representative power-on state for each cell using majority voting. For every target device, I extract the power-on state 51 times, which, by Bernoulli Trials, filters out (> 99%) the impact of noise. The system remains at a power-off state for 5 seconds between power cycles to discharge all the capacitance in the system, and all of my experiments extract power-on states at 25°C and a target’s nominal supply voltage.

The list of my target hardware includes a wide range of devices from different silicon vendors, including *NXP*, *Broadcom*, *Microchip*, and *Texas Instruments* (see Table 5.1). I test the targets for their power-on state accessibility, debugging techniques, and aging acceleration methods.

System-on-Chip	Core	SRAM size	TrustZone	Access to uncontaminated power-on state	Aging acceleration	Manufacturer
ATSAM11E16A [114]	ARM Cortex-M23	16KB	✓	✓	✓	Microchip
LPC55569JBD100 [122]	Dual-core ARM Cortex-M33	320KB	✓	✓	✓	NXP
M263KIAAE [37]	ARM Cortex-M23	96KB	✓	✓	✓	Nuvoton
M2351SFSIAAP [35]	ARM Cortex-M23	96KB	✓	✓	✓	Nuvoton
M252KG6AE [36]	ARM Cortex-M23	32KB	✓	✓	✓	Nuvoton
M251SD2AE [36]	ARM Cortex-M23	12KB	✓	✓	✓	Nuvoton
STM32L562 [163]	ARM Cortex-M33	40KB	✓	✓	✓	STMicroelectronics
BCM2837 (RPi3) [134]	Quad-core ARM Cortex-A53	L1:128KB, L2:512KB	✓	✓	✓	Broadcom
BCM2711 (RPi4) [135]	Quad-core ARM Cortex-A72	L1:320KB, L2:1MB	✓	✓	✓	Broadcom
R7FS1JA783A01CFM [44]	ARM Cortex-M23	32KB	✓	✓	✓	Renesas Electronics
MSP432P401 [69]	ARM Cortex-M4	64KB	✓	✓	✓	Texas Instruments
MSP430G2553 [71]	MSP430 single cycle	0.5KB	✓	✓	✓	Texas Instruments
EFM32WG990F256 [156]	ARM Cortex-M4	32KB	✓	✓	✓	Silicon Labs

Table 5.1: A list of devices that we verified are susceptible to our long-term data remanence attack.

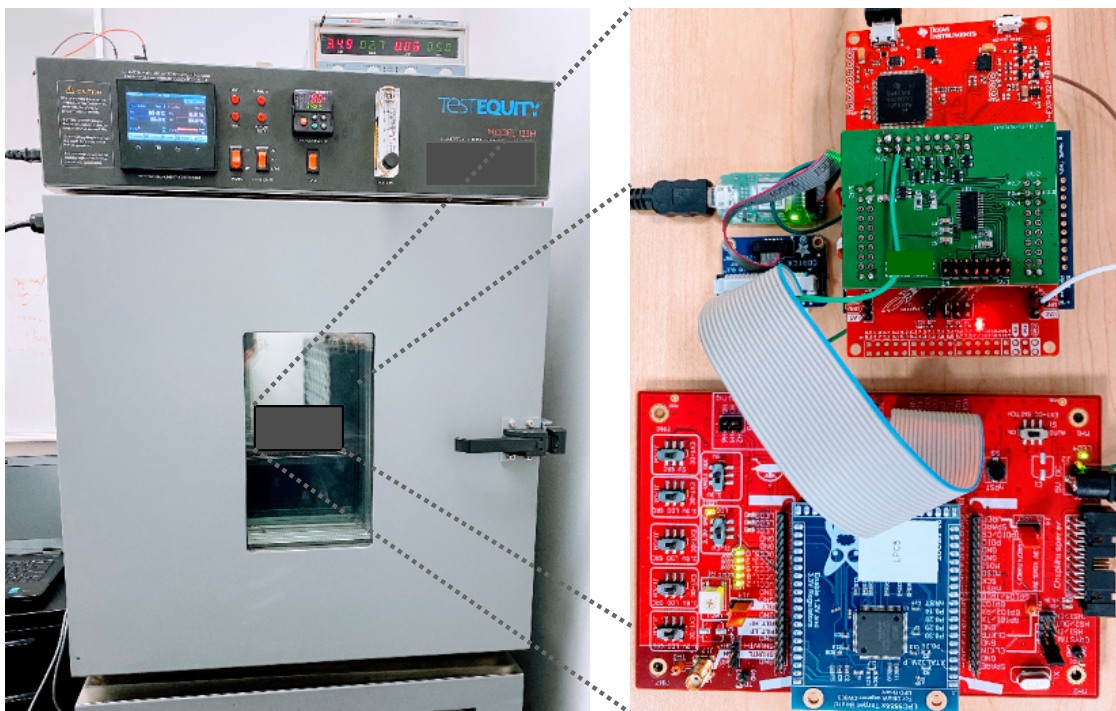


Figure 5.6: Pictures of my experimental setup. Detachable target hardware sits on a CW308 baseboard [68] (bottom right), which is connected to the power-control board and debug hardware (top right).

5.5.2 Tuning Stress Conditions to Enhance Accuracy

To find an acceleration condition that maximizes retrieved information accuracy, I study SRAM's response by filling three SAML11 [114] SoCs with *all 0s* (to allow static aging of cells) and exposing them to different aging conditions. Figure 5.7a shows an upward shift in the number of 1s because stress-induced degradation inverts the power-on state of a cell when I stress it with the power-on state stored (§5.2). Following a logarithmic trend, the $0 \rightarrow 1$ conversion rate (*i.e.*, aging-induced change) reduces over time. I observe an insignificant change in the number of 1s in the SRAM when it operates under nominal voltage and elevated temperature ($V_{core} = 1.2V$ and $T = 85^\circ C$), *but* elevated voltage ($V_{core} = 4.8V$) combined with high temperature ($T = 85^\circ C$) increases the acceleration significantly. Although SoCs allow operating temperatures higher than $85^\circ C$, I keep it at the standard industrial limit to stay within a realistic attack scenario as other components in the target system (*e.g.*, soldering) become vulnerable to damages beyond this temperature.

To understand how *strong* and *weak* cells behave under stress, I conduct another experiment with three SAML11 devices. I write all 1s to one chip and all 0s to another chip and expose them to stress ($V_{core} = 4.8V$ and $T = 85^\circ C$) for 16 hours, which allows us to observe how cells change when stressed with both logic states. The third chip contains its power-on state at nominal operating conditions (*i.e.*, no stress). Figure 5.7b plots the relation between pre and post-stress bias. The SRAM with a power-on state in it and nominal operating condition still contains *weak cells* (cells along the diagonal line). Chips that undergo accelerated aging have no weak cells present in both pre-stress and post-stress stages. When a weak cell undergoes stress, the data-directed aging aligns its power-on state with the complement of the data; in the following sections (§5.6, §5.7), I will show how this observation allows accuracy improvement in retrieved information.

We cannot complement the power-on state for all the cells: the relative power-on speed between the inverters (§5.2) can be so different that the aging-induced degradation fails to reverse the relation. This phenomenon introduces errors in information retrieval. Longer stress time improves the error performance of the attack, and to provide a better picture of how stress time affects the

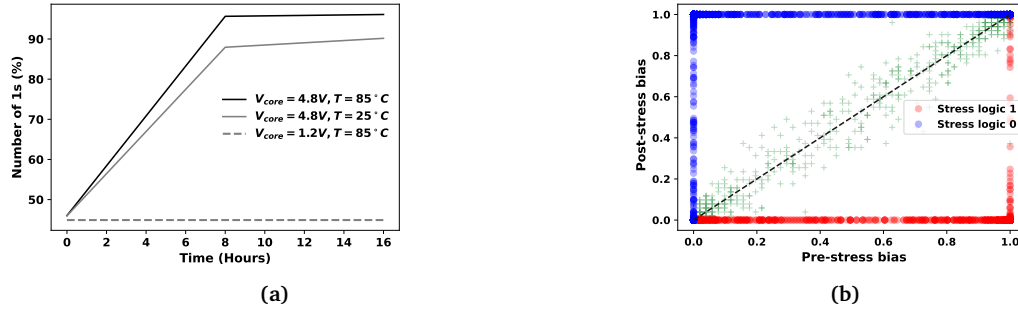


Figure 5.7: (a) Response of a SAML11 chip when exposed to stress at different stress conditions with extremely biased software (i.e., all 0's). (b) Stress response of cells with different logic states.

accuracy of retrieved data, Table 5.2 provides accuracy vs. stress time for LPC55S699 and SAML11 devices, respectively. *It is evident that longer stress time increases accuracy for both devices, although the rate of changes decreases over time.*

Stress time (hours)	8	16	24
Accuracy	77.34%	81.78%	87.99%

(a)

Stress time (hours)	2	4	8	16
Accuracy	88.23%	90.66%	95.67%	96.11%

(b)

Table 5.2: Effect of stress time on the retrieved information accuracy in (a) an LPC55S69 [122] device. and (b) a SAML11 [114] device.

As mentioned before, the magnitude of aging-induced degradation varies across device families. For example, I observe an 18% difference in accuracy between an LPC55S69 and SAML11 after 8 hours of stress at their respective acceleration voltages (§5.6 and §5.7), showing significant accuracy variation across device classes. When exposed to the same aging condition, the variation is much less within the same device class (but among different chips). This variation purely comes from the fabricated transistors' process variation, not on-chip or off-chip circuit designs. For example, I observed a 2% variation in the retrieved data accuracy on LPC55S69 devices when they go through the same stress condition (*i.e.*, voltage, temperature, and stress time).

The main takeaway is that, although the fundamental vulnerability of long-term data remanence

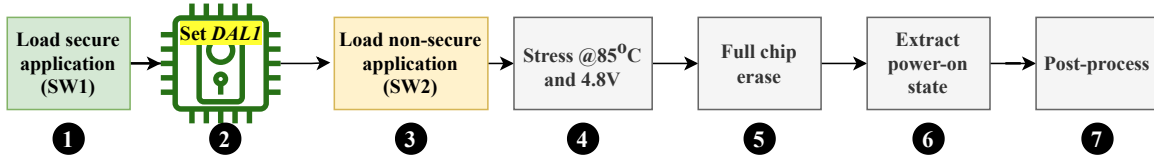


Figure 5.8: An attack model for *Secure World* AES key extraction.

remains unchanged, the accuracy of secret retrieval varies based on the manufacturing technology node and underlying circuit structure of the device. As a result, achieving maximum accuracy requires device-family-specific fine-tuning of stress variables, such as total stress time and acceleration voltage.

5.6 Attack #1: Exfiltrate an AES key from TrustZone

In this section, I examine a **threat model** where a non-secure user exploits SRAM's *long-term data remanence* to extract a secret key from the *Secure World*. For demonstration, I write a secure application that provides encryption/decryption services to the *Normal World* using NSC functions. The secure application receives encrypted texts from a non-secure application through NSC functions and decrypts them using an on-chip AES engine and a 128-bit secret key. This example emulates a communication scenario from outside of the chip where the *Normal World* is responsible for receiving information, performing error correction, and then forwarding the encrypted information to the *Secure World*. This scenario represents an extreme version of secure executions where TrustZone and on-chip cryptographic hardware guarantee the safety of all operations.

To implement the above attack, I use a device with TrustZone and an on-chip cryptographic accelerator; a SAML11 microcontroller [114] satisfies these requirements. This SoC has a Cortex-M23 core with TrustZone extension, and it features an array of security hardware, including an on-chip crypto-engine (CRYA) designed for IoT applications [115]. SAML11 devices allow setting three Debug Access Levels (DAL), which controls the information accessibly by a debugger. A device with DAL2 setting allows access to any part of the system. A secure user with DAL2 access programs the

Memory type	Base address	Size (bytes)	Security attributes
Flash	0x00000000	0x7C00	Secure
Flash	0x00007C00	0x0400	NSC
Flash	0x00008000	0x8000	Non-secure
SRAM	0x20000000	0x2000	Secure
SRAM	0x20002000	0x2000	Non-secure

Table 5.3: The memory map of my system. A secure application (SW1) uses the memory area marked as secure and NSC, whereas the non-secure area belongs to the non-secure application (SW2).

Stored data	Power-on state		Interpreted data	Correctness	% of bits	Transition type
	Pre-stress	Post-stress				
0	0	0	1	✗	2.19%	Flipping failure
0	0	1	0	✓	30.31%	Flipping success
0	1	1	0	✓	23.11%	Reinforcing
1	0	0	1	✓	26.41%	Reinforcing
1	1	0	1	✓	17.38%	Flipping success
1	1	1	0	✗	0.61%	Flipping failure

Table 5.4: The cell power-on state changes observed in AES key retrieval attack.

device and sets the debug access level to DAL1, which prevents the debugger from accessing the secure part of the system. A non-secure user, however, can escalate the debug access level by erasing the entire chip (both volatile and non-volatile memory). This feature prevents access to secure software's code or data without compromising the debugging features available to a non-secure user. Table 5.3 lists the memory map of my demonstration system.

Figure 5.8 illustrates the flow of the attack. A secure user with DAL2 access programs the secure software (SW1) in the device (❶) and then restricts the access level to DAL1(❷). At this stage, the device's secure memory is inaccessible to a non-secure application. A non-secure user programs software (SW2) that requests services from SW1 using NSC functions (❸). The non-secure user exposes the device to stress for 16 hours at an elevated voltage (4x) and temperature (3.4x)(❹). Then, the non-secure user erases the full chip to escalate the debug access level from DAL1 to DAL2 (❺) and extract power-on states of the SRAM (❻). Note that TrustZone protects the memory by erasing it before increasing a debug level. The final stage of the attack involves complementing each retrieved bit from SRAM and running a scenario-dependent error correction scheme to construct the key from the memory dump (❼).

Figure 5.9 plots the heatmaps of the pre-stress memory dump and retrieved information.² I locate different sections (e.g., global variables, stack, heap, etc.) of the secure software from the compiled binary and compare it with the retrieved data. **The accuracy of the retrieved data is 97.2%.**

Error source: The mean error in the retrieved words is 2.8%. I report that the words that compose the AES key contain no error; however, I consider the mean error in all of my further calculations to keep it applicable to most cases. By analyzing different types of pre-stress to post-stress changes in the power-on state, I investigate the error source (see Table 5.4). Data in an SRAM cell either reinforces the power-on state or weakens it by affecting the relative current driving strength of inverters (§5.2.2). Manufacturing variation can make an SRAM cell so strongly biased towards a logic state that stress-induced degradation fails to flip the relative speed of the inverters. This phenomenon leads to errors in post-stress information retrieval. The red lines in Figure 5.10 illustrate such transitions.

Key extraction scenario #1: I assume the most common case is when an attacker cannot access the device before a *secure user* programs it. That is, the attacker has no access to the pre-stress power-on values for a target device. In this case, the retrieved key is a corrupted version of the original key with a 2.8% error. An exhaustive key search within $n = 4$ bit Hamming distance ($128 \times 0.028 \approx 4 \text{ bit}$) of the original $N = 128$ bit key produces an *expected* key search space (L) $\approx 2^{23}$ (see Equation 5.1), which a modern computer covers in seconds.

$$L = \log_2 \left[\sum_{i=1}^n \binom{N}{N-i} \right] \quad (5.1)$$

Key extraction scenario #2: In an attack model where a non-secure user has access to a target before shipping it to the secure user for proprietary software programming, an attacker (i.e., the non-secure user) can further reduce the error using enrolled power-on states of a target device.

As discussed above, errors in the retrieved keys come from the cells that retain their pre-stress power-on state even after exposure to stress (see red lines in Figure 5.10). A probabilistic inter-

²I store two blocks of data, one with all 0s (black stripe) and the other with all 1s (white stripe), in the non-secure part of the memory to investigate whether there are any discrepancies between the imprinting process in the secure and non-secure area; my results indicate no difference.

pretation, instead of binary 1/0, of a cell's post-stress value improves the accuracy of the retrieved data. If a cell powers on at a defined logic state 100% of the time across trials at pre-stress and post-stress, it is impossible to correct the error generated from it. On the other hand, **pre-stress strong behavior and post-stress weak behavior reveal the cell's logic state during the stress**. These types of cells contribute to **45.25%** of the total erroneous interpretation: with enrolment power-on, I can correct this error. For example, let us assume a cell always powers on at 1 before stress (*i.e., strong 1 cell*). I expose it to stress, and the cell's power-on state becomes 1 and 0, 80% and 20% of time, respectively (*i.e., weak cell*). Such *weakening* of a cell's bias happens if it undergoes stress with its power-on state. Therefore, I interpret this cell contains 1 during stress as opposed to 0. I apply similar corrections to the *strong 0 cells* that become *weak cell* after stress. These considerations reduce the error from **2.8% to 1.27%**, and at this error at this level, the *expected* key search space comes down to $\approx 2^{13}$.

In addition to the SAML11 device, I launch this AES key exfiltration attack in a modern Cortex-A72 general-purpose processor in Section 5.9.

5.7 Attack #2: Exfiltrate Proprietary Firmware from Trust-Zone

SRAM provides lucrative benefits that motivate system designers to utilize them as a main memory for firmware execution. For example, cores execute instructions at a much faster speed when in on-chip SRAM compared to other common memories, such as Flash and DRAM. In addition, encrypting software while stored off-chip or Flash and decrypting it only in the on-chip memory prevents software piracy. Therefore, code executes faster and secures itself from low-effort physical attacks when in an on-chip SRAM.

Exposing proprietary firmware, my attack challenges the security of software that executes from an on-

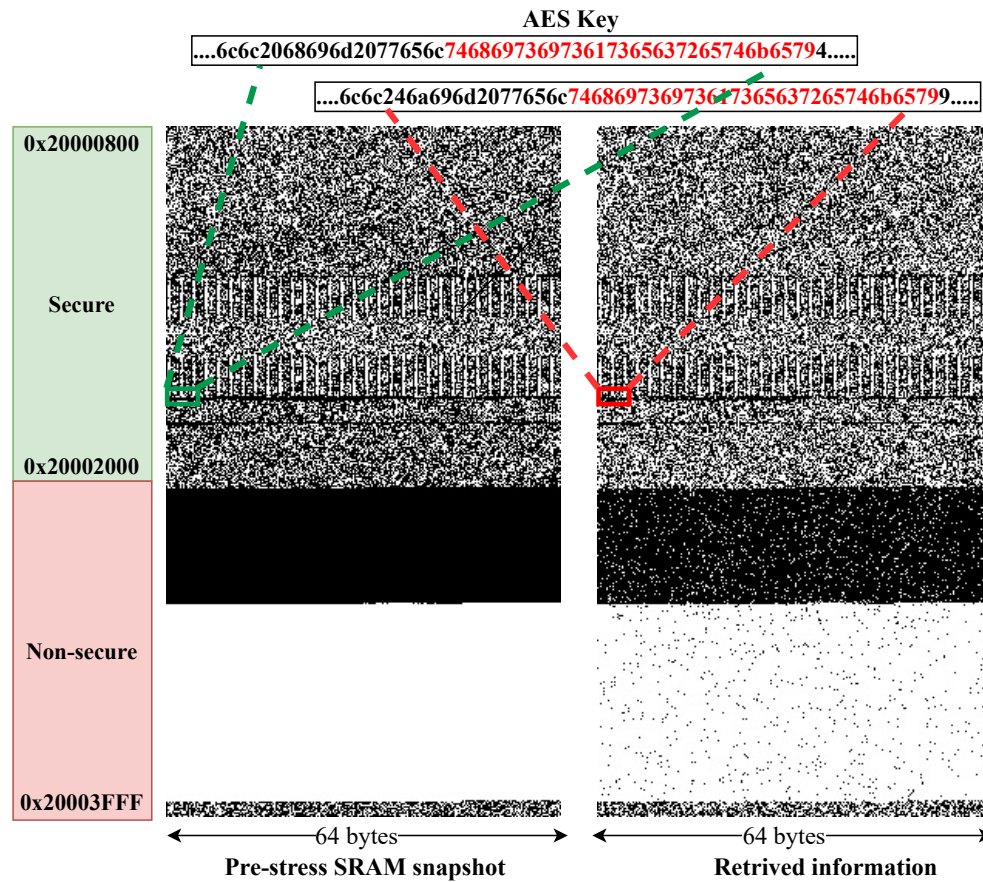


Figure 5.9: Heatmap of the pre-stress memory dump (left) and post-stress retrieved information (right). Note that the first 2KB of the SRAM is reserved for the on-chip ROM.

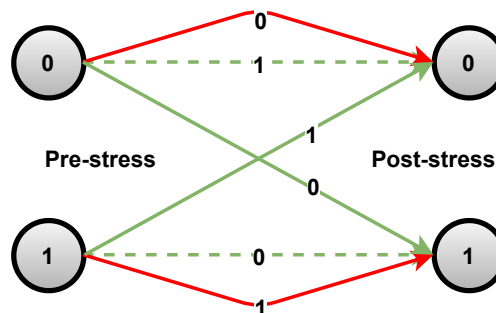


Figure 5.10: Data-directed aging response of an SRAM cell. When an SRAM cell stores the complement of its power-on state during stress, the post-stress power-on logic remains unchanged (dotted green lines). If it stores the same logic as its power-on state, the post-stress power-on logic flips (solid green lines). Contrary behavior (red lines) causes post-stress data interpretation errors.

chip SRAM. For demonstration, I use an LPC55S69 device, a dual-core Cortex-M33 microcontroller with TrustZone security extension, from NXP semiconductor. TrustZone and other advanced hardware security features, such as crypto accelerators and PUFs are part of CPU0, while CPU1 serves as a secondary core in support of CPU0.

Memory segments	Base address	Size (bytes)
CPU0 RAM	0x20000000	0x11000
CPU1 RAM	0x20012800	0x31800
Shared RAM	0x20011000	0x01800
CPU0 Flash	0x00000000	0xa0000

Table 5.5: Memory partitions between CPU0 and CPU1.

I design firmware for both cores, and they stay in a TrustZone-guarded secure Flash region. At startup, CPU0 initializes the system and loads the CPU1's firmware in the SRAM, which reduces the need for simultaneous Flash access by the cores, improving the system's security and performance. Once initialized, both cores start executing their respective firmware from two memories — Flash and SRAM. The entire memory region is marked as secure and CPU1 is the secure bus master; Table 5.5 lists the memory partitions for this experiment.

With a target to expose the TrustZone guarded firmware from the SRAM, I expose the device to stress by elevating voltage and temperature ($V_{core} = 5.5V$ and $T = 85^{\circ}C$) for 24 hours. Then, I erase the full chip to escalate debug level and extract the power-on state of the SRAM. I compare the retrieved binary of each 32-bit word (instruction length) with a pre-stress memory dump and plot the fractional Hamming distance in Figure 5.11. Note that the pre-stress memory dump is for analysis only, *i.e.*, it is unavailable to an attacker. As the firmware for CPU1 stays in the SRAM statically, the power-on states of the cells in that region align with the complement of the stored firmware bits, which leads to lower fractional Hamming distance (and error) in that area (0x20012A54 to 0x2001545C). The area where the fractional Hamming distance is around 0.5 contains a power-on state during stress. Table 5.6 shows the pre-stress and post-stress transitions among cells' power-on states.

Firmware bit	Pre-stress	Post-stress	Interpreted firmware bit	Correctness	% of bits			Transition types
					LPC1	LPC2	LPC3	
0	0	0	1	✗	7.80%	6.31%	7.08%	Flipping failure
0	0	1	0	✓	23.01%	22.23%	24.23%	Flipping success
0	1	1	0	✓	25.83%	28.04%	25.28%	Reinforcing
1	0	0	1	✓	22.23%	20.41%	23.51%	Reinforcing
1	1	0	1	✓	16.54%	15.93%	15.49%	Flipping success
1	1	1	0	✗	4.59%	6.97%	4.04%	Flipping failure

Table 5.6: Pre-stress and post-stress cell transitions of three LPC devices.

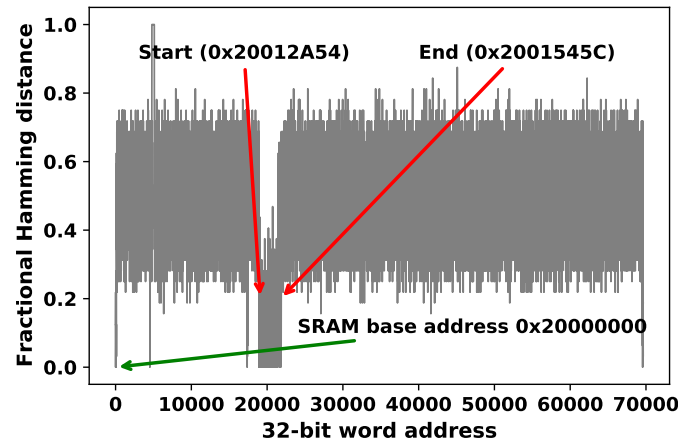


Figure 5.11: Post-stress fractional Hamming distance in each word (32-bit) of the SRAM (address base 0x20000000). *CPU1*'s firmware is within 0x20012800 to 0x2001545C, showing how firmware burns-in in the SRAM's power-on state.

	LPC1	LPC2	LPC3	Combined
Scenario # 1 accuracy	87.70%	86.70%	88.50%	95.82%
Scenario # 2 accuracy	93.20%	91.76%	93.36%	98.29%

Table 5.7: Firmware extraction accuracy for three target devices.

Attack scenario #1: The mean error in my 3-device experiment is 12.4% (see Table 5.7). Unlike keys, proprietary firmware is the same across devices, which allows an adversary to attack several devices running the same firmware and use majority voting on extracted firmware bits to reduce errors. The error in the retrieved data is largely random, which allows us to estimate the error with Bernoulli Trials in a scenario where an attacker has access to multiple devices. Theoretically, the error becomes 4.2% when the attacker has access to just three devices, each with $\approx 12.4\%$ random errors. my 3-device experiment retrieves the CPU1 firmware with $\approx 4.2\%$ error, which is the same as the expected theoretical result. I recreate $\approx 45\%$ of the machine code without applying any device or architecture-dependent noise filtering.

Attack scenario #2: Similar to Section 5.6, I consider an attack scenario where an attacker has access to devices' pre-stress power-on values. Access to enrolment data reduces errors in the retrieved firmware from individual target devices (see Table 5.7), which leads to much higher accuracy when combined with multi-device error correction. **The firmware extraction accuracy reaches 98.29%**, and at such high accuracy, I recreate 70.90% instructions error-free instructions—without applying more advanced hardware- or instruction-set-aware error correction.

An SRAM's stress-induced degradation shows spatial variation in the LPC55S69 device. Figure 5.12 plots the percentage of 1s in each 0.5KB block in the SRAM after a device goes through stress with all 0s. I observe different levels of aging responses at 64KB intervals. An LPC device has 5 SRAM blocks; the first four blocks are 64KB each, and the last block is 16 KB [122]. Therefore, firmware retrieval accuracy varies based on which SRAM bank it maps to—opening the door to the attacker forcing the placement of target code/data³.

This section shows how *UntrustZone* extracts proprietary firmware from a Cortex-M33's SRAM. Combining with the previous attack (§5.6), I demonstrate the attack's capability using a general-purpose processor, Cortex-A72 core, in Section 5.9.

³This scenario resembles *Flip feng shui* [137] attack where an attacker Virtual Machine (VM) allocates memory such that memory deduplication process merges an attacker page with victim VM's page, facilitating row hammering.

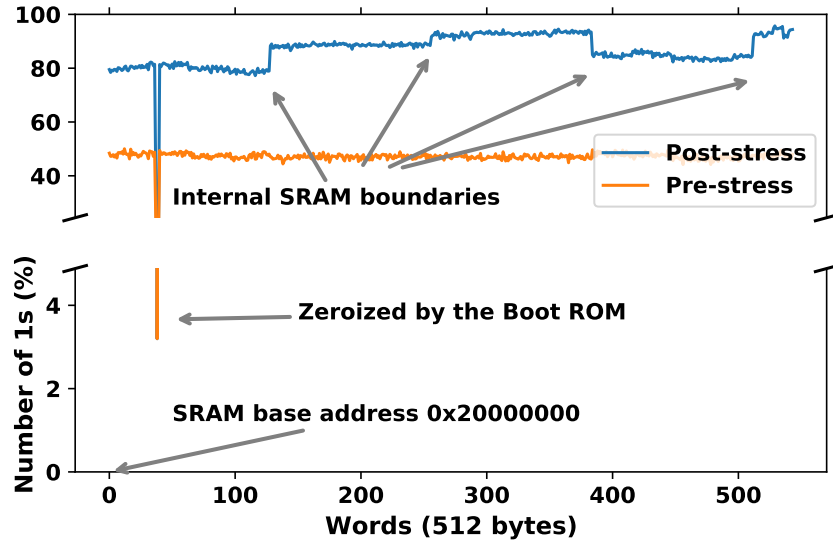


Figure 5.12: SRAM banks in the LPC device show different levels of stress-induced degradation based on their locations inside the die.

5.8 Attack #3: Exfiltrate Secrets from Cache

Many systems and architecture papers consider the *CPU cache* as the panacea against external memory disclosure attacks [32, 40, 190]. This notion stems from the fact that on-chip SRAM are safe from physical attacks, motivating encrypting external memory-bound transactions as a standard across trusted execution environment implementations. I show that long-term data remanence challenges this assumption by revealing secrets held even in the private cache of a CPU.

The ARM CPU’s cache lines are shared among the *Secure World* and *Normal World*: a *Secure World* application protects its cache line from *Normal World*’s application using an NS security flag. When a cache line carries a secure tag, the line is not available in any *Normal World* software, including the operating systems. My attack relies on the analog-domain changes of the constituent transistors of a cache, bypassing the security attribute of a cache line. First, I use a quad-core ARM Cortex-A53 processor (BCM2837) that comes with standard *Raspberry Pi 3 Model B(v1.2)* to demonstrate this attack. Then, I summarize all three attack scenarios with a Cortex-A72 processor.

5.8.1 Accessing Cache's Power-on State

Raspberry Pi (RPI) provides pre-compiled binary blobs that boot an embedded GPU core from external storage devices, *e.g.*, an SD card. It follows a boot chain that involves a combination of both on-chip and off-chip boot instruction sequences. First, the embedded GPU (*VideoCore IV*) boots from an on-chip ROM, which then loads a pre-compiled binary, `bootcode.bin`, in a shared (with ARM cluster) L2 cache. Second, it executes `start.elf` from an off-chip Non-Volatile Memory (NVM), which loads a kernel⁴ [133]. I introduce another stage of firmware before the ARM cores start executing a kernel and place it along with the pre-compiled binaries in the external NVM. my firmware ensures cores startup in aarch64 mode at EL3 exception level and utilizes CP15 coprocessor to extract L1 caches at power-on regardless of the cache lines' validity. CP15 instructions such as `msr s3_3_c15_c4_1, <Xd>` and `msr s3_3_c15_c4_0, <Xd>` allow reading i-cache and d-cache by iterating over ways, sets, and offsets [11]. Here, `Xd` is a 64-bit general CPU register.

A Cortex-A53 allows invalidating cache lines by executing IC IVAU and DC ISW instructions for i-cache and d-cache, respectively. Invalidation prevents a CPU from accessing a line but does not reset the cached data in data RAM. For example, IC IVAU instruction invalidates the entire L1 i-cache by resetting all i-cache tags to `0x7FFFFFFF`. However, the data in the cache lines remain unmodified, but the processor ignores them (*i.e.*, no cache hit).

I write a tool that programs and controls the core through *openocd* [136] and *JTAG interface*. Cache's power-on state extraction tool performs the following tasks:

1. Disables L1 memories (*e.g.*, caches, TLBs,) and makes the core busy-wait (*e.g.*, `spin: b spin`) to eliminate cache's power-on state contamination and to avoid hitting any undesired exceptions, respectively.
2. Establishes JTAG communication and halts all cores.
3. Loads a *cache extraction software*⁵ in the main memory (*i.e.*, *SDRAM*). This software reads

⁴The source codes for `bootcode.bin` and `start.elf` are confidential, and I do not have access to them.

⁵The software is in assembly (aarch64) to tightly control the execution.

the contents by iterating over the entire target cache and dumps it in the *SDRAM*.

4. Transfers the *SRDAM* contents to a debug host (e.g., PC) through *JTAG* interface.
5. Repeats steps 3 and 4 for all the target cores and caches.

5.8.2 Attacking ARM Core's L1 Cache

To apply accelerated stress on a device, I need access to the V_{core} pin(s). The detailed schematic of the Raspberry Pi boards and pin description of BCM2837 SoC are not publicly available. However, a closer examination on public RPi3 reduced-schematic [45] reveals that an RT8088A [138] switching regulator supplies 1.2V to the ARM cores and internal SRAM; this a typical power management scheme in almost any modern system [104]. The output of the switching regulator passes through an inductor (L3) before reaching the core voltage pins (§5.4). I remove this inductor from the board (while maintaining the live system voltage externally), which disconnects the switching regulator from supplying power to the SoC. A probe connected at the test pad PP58 of RPi3 from my power-control board manages the voltage level at the SoC's V_{core} , which allows us to elevate the voltage supplied to the SRAM.

For attack demonstration, I fill out the caches of all the cores with preset instructions fetched from the *SDRAM* and expose the system to stress. **At an acceleration voltage $V_{core} = 2.2V$ and $T = 85^{\circ}C$, I extract contents of an i-cache with 79.2% accuracy after 120 hours of stress.** Since my *cache extraction software* reads out *invalid* cache lines at startup, it bypasses the TrustZone-enforced security boundary.

5.9 Unifying the Attack Scenarios

Consolidating the attack scenarios into one system, I summarize *UntrustZone* using a newer ARM SoC, Broadcom BCM2711, which has a quad-core Cortex-A72 CPU (aka Raspberry Pi 4) [135]. To

show the generality of *UntrustZone* attack by combining all the attack scenarios described above with an overarching threat model that assumes secret data (attack #1) and proprietary software (attack #2) are in the on-chip cache (attack #3).

The first step in the *UntrustZone* deployment phase is to identify the acceleration voltage pin that is located near the power management integrated circuit (*MXL7704*); this chip drives the V_{core} pin (TP15 pad in the PCB [128]). As described in section 5.5.1, I determine the acceleration voltage with guidance from the datasheet [128] and experimentation. While the software and toolchain needed for power-on extraction remain the same as BCM2837 (§5.8), the acceleration voltage is 1.62V (2.025× nominal voltage).

At attack deployment, the victim cores 0 and 1 run on-chip encryption/decryption using a 128-bit AES engine; this AES does not use memory access once loaded into the L1 cache due to its implementation using vector registers and on-demand calculation of the sub-keys and other intermediate states.⁶ Encryption/decryption operations using this AES never save intermediate states in the off-chip memory, making it secure against external memory attacks, such as cold boot [59]. It is essentially the same as attack #1 (§5.6), where the cryptographic service is provided by the on-chip secure state of a processor. The other two cores execute typical software operations while keeping both software and data in the cores' L1 private cache, making it inaccessible from the RAM in plain text (*i.e.*, attack #2).

The AES key extraction accuracy reaches **93.2% after 120** hours of aging, and I observe similar results in the other two cores' instruction (93.3%) and data (93.4%) extraction. Similar accuracy across different cores and cache types is expected as they share the same power bus, hence experience the same accelerated stress. The mean accuracy of all four cores (both i-cache and d-cache) reaches ~ 95% after 168 hours of stress.

⁶The implementation is *stackless* and in assembly, which allows doubt-free post-stress error calculation.

5.10 Countermeasures

Long-term data remanence is an analog-domain attack that exploits device-level changes to leak on-chip secrets. A viable defense needs careful consideration of both software and hardware avenues. We provide a qualitative analysis of the possible defense mechanisms and their expected implications on the system's design and performance.

5.10.1 Software Defenses

Initializing the SRAM at startup

SRAM's power-on state is the information carrier for this attack as it provides digital access to the data-directed changes in the analog domain. Therefore, initializing the entire SRAM as a part of the secure boot process prevents *UntrustZone* attack. This type of countermeasure come with two potential downsides:

- SRAM initialization as part of CPU power-up sequence stretches startup time by a few hundred cycles to thousands of cycles. For example, an LPC [122] device at its highest boot speed (96MHz) takes $640\mu s$ to initialize its 320KB SRAM (61440 cycles), which is a significant delay for embedded applications.
- Initializing an SRAM's power-on state prevents security algorithms (*e.g.*, TRNGs [63], PUFs [108, 140, 188], recycled IC detection [56]) from utilizing hardware uniqueness and randomness at run-time.

Instead of using CPU cores to reset, a more flexible and efficient option is to utilize self-test engines (*i.e.*, MBIST) to reset the SRAMs with an option to configure it according to a system designer's need.

Scrambling the SRAM data at run-time

Balancing the stress between inverters in an SRAM cell is a potential software solution that prevents imprinting data in the aging-induced asymmetry. This scheme requires active run-time support where software periodically inverts or scrambles the stored bits [100]. Our attacks drive the core voltage well beyond the nominal, which freezes a processor's execution, making this software defense ineffective.

5.10.2 Hardware Defenses

Preventing aging acceleration

Aging acceleration is the key to launching *UntrustZone* attack, which comes from thermal and electrical stress. Averting thermal acceleration is relatively simpler and can be done with help from modern processors' temperature sensing and management hardware, such as Thermtrip [1] and Frequency throttling. Thermtrip in Intel CPU prevents damaging processors from operating under excessively high temperatures [1], and frequency throttling is common across all high-performance CPUs to keep the devices under their thermal limits. Orienting these mechanisms to detect adversarial aging and erasing sensitive information upon such detection can be a solution to eliminate the thermal component of the acceleration. Given that a device's thermal operating condition varies depending on its applications, a *combined* software and hardware solution is necessary to provide configurable solutions to thermal stress tolerance.

Electric stress is the driving factor in acceleration and is much harder to prevent (§5.2.3). A device needs to enforce strict operating voltage without any error margin, which is impractical because it requires modeling a power distribution network's behavior well before manufacturing. Process variations, the unpredictability of run-time noise, and the need for multiple operating modes complicate such a strict nominal voltage limit, making most devices respond to accelerated aging to

some degree. We observe the following two types of behaviors in target devices that limit accelerated aging using voltage:

- **Bypassing the excess energy through circuits before reaching the core:** As a standard design practice, pins in a chip include measures for over-voltage (*e.g.*, ESD) and reverse-polarity protection. A chip's behavior in these circumstances depends on the design and implementation of the underlying circuitry. For instance, a BCM2837 draws $> 2.5\text{A}$ current when core voltage V_{core} goes beyond 2V (1.67 $\times$). Such high current consumption indicates an over-voltage prevention mechanism bypasses excess power to the ground by clipping off the excess voltage.
- **Erasing the volatile memory contents:** A hardware measure that resets the CPU and memory beyond recommended voltage is also observable in devices. In experimentation with M2351SFSIAAP [35], I observe that the device resets the I/O registers and SRAMs if the voltage rises beyond 2.3V in its core voltage pin (LDO_CAP)⁷.

While these protection mechanisms prevent over-voltage, they kick in only when the voltage crosses the design margin, which is higher than the operating voltage limit. It reduces the acceleration, but longer stress time compensates for the slower acceleration.

Restricting debug access

The proposed attack requires debug access or loading a modified kernel (after resetting TrustZone and deleting the contents of the memory in Cortex-M devices) to read out uninitialized SRAM values from secure memory area. Debug authentication features [121] or mandating a trusted boot process should thwart reading out imprinted SRAM data, ultimately preventing the attack. Disabling debug permanently complicates return material analysis and software update process in

⁷The voltage at this pin is 1.2V when operating in Normal mode (48MHz) and 1.26V in Turbo mode (64MHz), which leaves a room for accelerating a device's aging.

lower-end devices, which are essential methods to analyze a system failure and product improvement [149]. Note that there are attacks that bring the device to its factory reset condition or circumvent the disable debug settings [79, 167].

5.11 Discussion

Our experiments with a range of SoCs uncover *a general trend* among different devices that I expect to hold for relatively more complex processors employed by desktop- and server-class systems. First, an increasing number of devices— including desktop and server-class processors— allow access to their caches directly either through a memory-mapped interface (*e.g.*, RISC-V [155]) or co-processor interface (*e.g.*, Cortex-A53 [12], Cortex-A78 [93], Cortex-A72 [92]). Such architectural flexibility and feature offer several advantages, such as the ability to execute software quickly from on-chip memory, ensuring run-time security against off-chip memory attacks, and facilitating debugging features for low-level memory translation issues.

Second, the requirement for a robust and low-noise power supply from external sources becomes essential as hardware complexity increases. Current fluctuation in complex processors (*e.g.*, Cortex-A53) is significantly more compared to embedded devices. Therefore, I expect complex processors' power delivery systems to be similar to the device I assessed in our experiments.

Third, desktop- and server-class processors allow overclocking by elevating the supply voltage. This means the door is open to using this access for accelerated imprinting on other processors. For example, Intel Core and Intel Xeon E processors allow operating at a nominal 0.9V, but they allow voltage overdrive to at least 1.52V [72].

Fourth, voltage scaling lags behind geometric scaling, which increases the electric field density in a device, making smaller technology nodes more susceptible to aging-induced degradation. This trend leads to increased information leakage from SRAM manufactured using newer technology nodes. For example, the accuracy of imprinted and retrieved information from a 40nm LPC de-

vice [43] is over 4x more than that of a 130nm [112] MSP430 [69], even when the LPC device spends less than half the time having software imprinted into SRAM’s analog domain.

Considering the attack enablers outlined in Section 5.4 and the observed trends that extend to desktop- and server-class processors, I anticipate that our attack methodology will be applicable to those devices as well. Our analysis of Intel SGX and AMD SEV indicates that these more sophisticated TEEs need further experimental evaluation to determine the impact of *UntrustZone* attack.

5.12 Related work

Given the popularity of ARM devices, there is a significant research drive to evaluate the security of ARM TrustZone.

5.12.1 Exploiting Voltage and Frequency Management Hardware

CLKSCREW [164] and *VoltJockey* [130] manipulate the energy management systems shared among multiple cores to push it beyond the vendor recommended limit. *CLKSCREW* manipulates the frequency, and *VoltJockey* exploits the voltage from a non-secure kernel driver to induce timing faults in the secure cores so that it produces incorrect computations. The authors show how an attacker bypasses RSA-based signature verification and loads an *untrusted* application into the TrustZone. These attacks apply only to multi-core CPUs with shared dynamic voltage and frequency management systems.

5.12.2 Exploiting Cache and DRAM

Secure World and *Normal World* share on-chip memory such as caches and Branch Target Buffer (BTB). Cache allocation and eviction are not restricted by the security attribute of a cache-line.

That is, hardware does not restrict a *Normal world's* application from evicting a cache-line that belongs to *Secure World*. While this design decision optimizes performance, it leads to numerous cache-contention-based side-channel attacks such as *ARMageddon* [95], *TruSpy* [191], *Cache storage channels* [52], and *Prime+Count* [29]. Since the Branch Target Buffer is a shared resource between the *Secure* and *Normal World*, cache-contention-based attacks also apply [144]. BTB-based exploits increase the spatial resolution of the probing mechanism because the BTB stores information at byte granularity rather than cache-line granularity [25].

The Rowhammer attack also impacts ARM TrustZone-enabled devices [23], where a malicious kernel from the *Normal World* generates high-rate memory read requests close to a secure memory boundary. These high-rate read requests corrupt an RSA private key in the secure memory (*i.e.*, DRAM), leading to faulty signature generation in the *Secure World*. The comparison of the message and signature in the *Normal World* leaks the RSA private key. Unlike shared-on-chip-buffer-based side-channel attacks, keeping secrets on-chip defeats Rowhammer attacks.

5.12.3 Exploiting Configurable Hardware

AMBA AXI interface of TrustZone-enabled devices contains a security flag (NS bit) for *read* and *write* channels in the main system bus. This bit carries the security state of the CPU. Reconfigurable hardware such as FPGA (*e.g.*, Zynq-7000 [184]) can be connected to the system's main bus to allow flexible and efficient hardware/software co-design. Benhani *et al.* show that such heterogeneous hardware architectures are security risks because malicious hardware in the FPGA fabric potentially breaks ARM TrustZone security [109]. Jacob *et al.* show that malicious hardware in the FPGA can interfere with the SOC's secure boot process, allowing a CPU to load authorized kernel [73].

5.13 Conclusion

In this chapter, I show how long-term data remanence is a threat to hardware-backed secure computing environments. Our attack shows, contrary to the assumptions of previous defenses, keeping secrets on-chip—even when guarded by the hardware-level protections of a Trusted Execution Environment—is insufficient to keep them secure when an attacker has physical access to the chip. I implement and validate the long-term data remanence vulnerability on a range of devices spanning complexity, cost, performance, age, and manufacturer. I then use the capability to carry out three attacks. The attacks steal secret code and data from TrustZone-protected main memory and on-chip cache lines. Our results show how an attacker can efficiently imprint software secrets into SRAM’s analog domain and reliably retrieve those imprinted secrets through measurements of SRAM’s power-on state. Our analysis of the defensive trade-space suggests that a robust solution requires hardware and low-level software modification.

Beyond exposing a universal weakness in secure computing devices, this paper highlights a broader threat: transistor-aging-based side-channels. While the circuits community has long been aware of the effects of transistor aging, their focus has been on its performance and reliability impacts, not security. This paper shines a light on the ramifications of security of data-dependent transistor aging. I believe this paper will inspire future researchers to uncover other security issues related to transistor aging and new side-channels that leverage the latent information left behind by data-dependent transistor aging.

Chapter 6

Conclusion and Future Work

The prevalence of hardware-level vulnerabilities, covering micro-architectural to analog domain weaknesses, opens up a wider threat space to overall system integrity. Considering a symbiotic relationship between software and hardware security is paramount in safeguarding systems from potential breaches. This dissertation represents a pivotal stride in advancing system security by exploring low-level circuit behavior and its security implications.

I show that retaining the internal SRAM states in modern SoCs is challenging with cold boot attacks at extremely low temperatures, as previously suggested. This dissertation introduces a method to simulate cold boot conditions through external voltage probing of on-chip SRAM. In this work, I highlight that hardware designers need to consider architectural design choices that may have security implications. Additionally, I develop a technique for hiding information within the SRAM's analog layer, laying the groundwork for secure, plausibly deniable communication channels. This involves employing accelerated aging to cause wear in SRAM cells, allowing for information encoding in the analog domain, and preventing authorized scrutiny of sensitive data

While traditional cold boot data retention strategies are ineffective against hardware-backed security environments like ARM's TrustZone, I propose a new attack capable of extracting secrets from on-chip SRAM under such strict security isolation. This attack is feasible due to the shared physical circuit between secure and non-secure states, enabling unauthorized access to secrets by retaining information from the secure state.

6.1 Future work

The increasing demand for high-performance integrated circuits has led to the widespread use of advanced manufacturing technologies, such as those based on nanoscale fabrication processes. This trend brings us scores of advantages, such as higher computing power at the same cost of production and a similar power budget. The miniaturization of transistors also increases susceptibility to aging and degradation, negatively impacting device performance and reliability over time. Aging deviates properties of transistors from their intended behavior, causing them to produce faulty computation to downright failure. As my previous work shows [103, 105], the unreliability problem becomes a security problem when an adversary introduces malicious aging– a process where the aging of a circuit is intentionally accelerated, leading to its premature failure or faulty computation targeting security components [77]. This can occur through various means, including voltage and temperature stress, ionizing radiation, and electromagnetic interference.

The impact of malicious circuit aging is not limited to attacking personal devices, as it can have significant economic and security implications at the societal level. At the individual level, malicious circuit aging can result in the loss of critical data, financial loss, and even physical harm. At the societal level, malicious circuit aging can lead to disruptions in critical infrastructure, such as transportation and communication systems. Therefore, understanding the economic and security impact of malicious circuit aging is critical for developing effective strategies to mitigate its effects. My ongoing research will pursue two main directions:

6.1.1 Malicious Software-induced Aging Faults

Transistor aging has several implications in terms of affecting system performance, security, and reliability. One of the most significant implications of transistor aging is the shift in the propagation delay of circuit components, which can lead to timing violations and potential functional failures if unchecked [87]. As transistors age, their threshold voltages shift, which can cause circuit compo-

nents, such as registers and pipelines, to operate at slower speeds than intended, which may lead to timing violations in the critical paths of the system, causing intermittent and permanent functional failures. From a security side perspective, aging can distort Physical Unclonable Functions (PUF) response, clone the entire PUF [142], fault injection [77], and reveal a chips usage even if the package is new [41, 54].

While aging is a natural phenomenon—which is typically a slow process spanning years—the underlying physics of aging allows accelerated stress, which leads to numerous security implications such as steganography design [103] and data retention attacks [176]. The fundamental idea of accelerated aging is that elevated voltage and temperature make devices wear out faster. Typically, device performance degradation is examined as a possible attack vector when an adversary has physical access to the device. My previous work aligns with a similar theme [103, 105], where one needs to have access to a device for a certain amount of time in order to exploit aging-based vulnerability. This future work switches gears and aims to look at device aging from a security perspective when an adversary maliciously ages a device to cause security issues from *only* software access.

The current research focuses on the security risks of physical attacks on devices and does not address remote aging attacks, due to the following challenges:

- **Challenge 1:** Without physical access to a device, what is an effective way to push processors into a state that maximizes aging-induced degradation of its underlying circuit?
- **Challenge 2:** What is a time-efficient way to test hardware to quantify aging-induced vulnerability?

The high-level approach to this work is to craft a set of instructions that drives a processor’s internal circuit into a state where an adversary can target on-chip security critical assets for malicious aging. The primary objectives of such attacks include activating hardware Trojans, compromising competitors’ hardware in cloud environments, and weakening privilege checks. To create software that induces malicious aging over time, attempting every combination of instruction and input in a typical hardware simulation is impractical. Instead, I plan to utilize newer techniques known as

hardware fuzzing, where hardware models are converted to software simulation models [168]. This way, a targeted set of instructions can be found by harnessing the power of high-speed software testing methods.

Despite having an effective instruction set, aging as a process is slow and impractical for execution on a remote machine within a timeframe that could be considered reasonable or profitable. This introduces another research problem: finding methods to accelerate the aging process in a remote system. One approach to expedite aging, controlling the system's low-level voltage interface, is not available to an attacker, as discussed in Chapter 4. However, it is feasible to increase the device's temperature via the software interface. This can be achieved by either designing a specific sequence of instructions that generate high activity levels or by leveraging the on-chip circuitry's structural properties, such as inducing read/write collisions in SRAM with dual-port access.

6.1.2 SRAM Data Remanence-based Physical Attack Mitigation

To counter the physical attacks detailed in this dissertation, a comprehensive defense strategy encompassing both software and hardware measures is essential. In Chapters 3 and 5, I have qualitatively discussed potential mitigations for SRAM data remanence-based attacks. Future work focuses on the quantitative assessment of these defense strategies to develop an optimized solution that considers cost, performance, and security trade-offs.

As we have seen in the previous chapters, successful execution of these attacks involves two main phases:

- Forcing a victim device into a state where it becomes susceptible to data remanence.
- Extracting the retained secrets using a software interface.

Blocking data retention directly is challenging due to the need for hardware design changes and board-level modifications, which are expensive or unfeasible for existing systems. On the other

hand, preventing unauthorized access to the contents of the SRAM is a relatively more viable strategy. Consequently, defenses against these attacks should primarily focus on preventing the extraction of retained information.

Blocking direct access to the cache lines from co-processors' interface presents challenges, especially regarding boot speed and debugging of low-level memory problems, but is vital for enhancing system security. Additionally, the initial power-on state of SRAM serves as an entropy source for detecting counterfeit chips [55, 182], and restricting access to this data could hinder these innovative uses of SRAM. To effectively counteract data remanence attacks, a strategy involving the erasure of SRAM contents prior to the CPU transitioning control to external software is critical. Incorporating code within the boot ROM to erase SRAM prevents adversaries from accessing residual states. However, the process of erasing SRAM at power-on introduces a time delay (due to reduced boot speed), during which attackers have the opportunity to exploit various faults, such as voltage and clock glitching to bypass this security measure. Therefore, a detailed analysis of these trade-offs is crucial for formulating robust defenses against the attacks discussed in earlier chapters.

Bibliography

- [1] <https://www.intel.la/content/dam/www/public/us/en/documents/datasheets/pentium-dual-core-desktop-e2000-datasheet.pdf>. Accessed: 2023-7-15.
- [2] Muhammad Ashraful Alam and Souvik Mahapatra. A Comprehensive Model of PMOS NBTI Degradation. *Microelectronics Reliability*, 45(1):71–81, 2005.
- [3] Esteve Amat, E Amatllé, Sergio Gómez, Nivard Aymerich, Carmen G Almudéver, Francesc Moll, and Antonio Rubio. Systematic and Random Variability Analysis of Two Different 6T-SRAM Layout Topologies. *Microelectronics Journal*, 44(9):787–793, 2013.
- [4] Behnam Amelifard and Massoud Pedram. Optimal design of the power-delivery network for multiple voltage-island system-on-chips. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(6):888–900, 2009.
- [5] Aidin Ameri and Daryl Johnson. Covert channel over network time protocol. In *Proceedings of the 2017 International Conference on Cryptography, Security and Privacy*, pages 62–65, 2017.
- [6] Nikolaos Athanasios Anagnostopoulos, Tolga Arul, Markus Rosenstihl, André Schaller, Sebastian Gabmeyer, and Stefan Katzenbeisser. Low-temperature data remanence attacks against intrinsic sram pufs. In *2018 21st Euromicro Conference on Digital System Design (DSD)*, pages 581–585. IEEE, 2018.
- [7] Nikolaos Athanasios Anagnostopoulos, Tolga Arul, Markus Rosenstihl, André Schaller, Sebastian Gabmeyer, and Stefan Katzenbeisser. Attacking sram pufs using very-low-temperature data remanence. *Microprocessors and Microsystems*, 71:102864, 2019.

- [8] Ross Anderson and Markus Kuhn. Low Cost Attacks on Tamper Resistant Devices. In *International Workshop on Security Protocols*, pages 125–136. Springer, 1997.
- [9] ARM Limited. ARM Cortex-A Series: Programmers Guide for ARMv8-A.
- [10] Arm Limited. Security in ARMv8-A systems, 2017. <https://developer.arm.com/documentation/100935/0100/The-TrustZone-hardware-architecture-?lang=en>.
- [11] ARM Limited. Armv8-M Architecture Reference Manual, 2020. <https://developer.arm.com/documentation/ddi0553/bn/>.
- [12] ARM limited. ARM Cortex-A53 MPCore Processor Technical Reference Manual, 2021. <https://developer.arm.com/documentation/ddi0500/j/Introduction/Product-documentation-and-design-flow/Documentation>.
- [13] ARM Limited. L220 Cache Controller Technical Reference Manual, 2021. <https://developer.arm.com/documentation/ddi0329/l/functional-overview/functional-operation/trustzone-support-in-the-cache-controller>.
- [14] Wikipedia Article. Key disclosure law, 2020. https://en.wikipedia.org/wiki/Safety_of_journalists.
- [15] ASU. Predictive Technology Model (PTM), 2020. <http://ptm.asu.edu/>.
- [16] Marta Bagatin, Simone Gerardin, Alessandro Paccagnella, and Federico Faccio. Impact of NBTI aging on the single-event upset of SRAM cells. *IEEE Transactions on Nuclear Science*, 57(6):3245–3250, 2010.
- [17] Austen Barker, Staunton Sample, Yash Gupta, Anastasia McTaggart, Ethan L Miller, and Darrell DE Long. Artifice: A deniable steganographic file system. In *9th {USENIX} Workshop on Free and Open Communications on the Internet ({FOCI} 19)*, 2019.
- [18] Johannes Bauer, Michael Gruhn, and Felix C Freiling. Lest we forget: Cold-boot attacks on scrambled ddr3 memory. *Digital Investigation*, 16:S65–S74, 2016.

- [19] Erik-Oliver Blass, Travis Mayberry, Guevara Noubir, and Kaan Onarlioglu. Toward robust hidden volumes using write-only oblivious RAM. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 203–214, 2014.
- [20] Hristo Bojinov, Daniel Sanchez, Paul Reber, Dan Boneh, and Patrick Lincoln. Neuroscience meets cryptography: designing crypto primitives secure against rubber hose attacks. In *Presented as part of the 21st {USENIX} Security Symposium ({USENIX} Security 12)*, pages 129–141, 2012.
- [21] Alain Bravaix, Chloé Guérin, Vincent Huard, David Roy, Julien-Marc Roux, and Emmanuel Vincent. Hot-carrier Acceleration Factors for Low Power Management in DC-AC Stressed 40nm NMOS Node at High Temperature. In *International Reliability Physics Symposium*, pages 531–548. IEEE, 2009.
- [22] Cagla Cakir, Mudit Bhargava, and Ken Mai. 6t sram and 3t dram data retention and remanence characterization in 65nm bulk cmos. In *Proceedings of the IEEE 2012 Custom Integrated Circuits Conference*, pages 1–4. IEEE, 2012.
- [23] Pierre Carru. Attack TrustZone with Rowhammer, 2017. https://grehack.fr/data/2017/slides/GreHack17_Attack_TrustZone_with_Rowhammer.pdf.
- [24] Aniello Castiglione, Kim-Kwang Raymond Choo, Michele Nappi, and Stefano Ricciardi. Context Aware Ubiquitous Biometrics in Edge of Military Things. *Cloud Computing*, 4(6):16–20, 2017.
- [25] David Cerdeira, Nuno Santos, Pedro Fonseca, and Sandro Pinto. SoK: Understanding the Prevailing Security Vulnerabilities in TrustZone-assisted TEE Systems. In *Symposium on Security and Privacy (SP)*, pages 1416–1432. IEEE, 2020.
- [26] Ellick M. Chan, Jeffrey C. Carlyle, Francis M. David, Reza Farivar, and Roy H. Campbell. BootJacker: compromising computers using forced restarts. In *Proceedings of the 15th ACM*

- conference on Computer and communications security - CCS '08*, page 555, Alexandria, Virginia, USA, 2008. ACM Press. ISBN 978-1-59593-810-7. doi: 10.1145/1455770.1455840. URL <http://portal.acm.org/citation.cfm?doid=1455770.1455840>.
- [27] Bing Chang, Zhan Wang, Bo Chen, and Fengwei Zhang. Mobipluto: File system friendly deniable storage for mobile devices proceedings of the 31st annual computer security applications conference. *Los Angeles, CA, USA, December*, 2015.
- [28] Srivatsan Chellappa and Lawrence T Clark. SRAM-based Unique Chip Identifier Techniques. *Transactions on Very Large Scale Integration (VLSI) Systems*, 24(4):1213–1222, 2015.
- [29] Haehyun Cho, Penghui Zhang, Donguk Kim, Jinbum Park, Choong-Hoon Lee, Ziming Zhao, Adam Doupé, and Gail-Joon Ahn. Prime+ Count: Novel Cross-world Covert Channels on ARM Trustzone. In *Proceedings of the 34th Annual Computer Security Applications Conference*, pages 441–452, 2018.
- [30] Dawei Chu, Yuewu Wang, Lingguang Lei, Yanchu Li, Jiwu Jing, and Kun Sun. Ocram-assisted sensitive data protection on arm-based platform. In *European Symposium on Research in Computer Security*, pages 412–438. Springer, 2019.
- [31] Patrick Colp, Jiawen Zhang, James Gleeson, Sahil Suneja, Eyal De Lara, Himanshu Raj, Stefan Saroiu, and Alec Wolman. Protecting data on smartphones and tablets from memory attacks. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 177–189, 2015.
- [32] Patrick Colp, Jiawen Zhang, James Gleeson, Sahil Suneja, Eyal De Lara, Himanshu Raj, Stefan Saroiu, and Alec Wolman. Protecting Data on Smartphones and Tablets from Memory Attacks. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 177–189, 2015.
- [33] Patrick Colp, Jiawen Zhang, James Gleeson, Sahil Suneja, Eyal de Lara, Himanshu Raj, Stefan Saroiu, and Alec Wolman. Protecting Data on Smartphones and Tablets from Memory

- Attacks. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 177–189, Istanbul Turkey, March 2015. ACM. ISBN 978-1-4503-2835-7. doi: 10.1145/2694344.2694380. URL <https://dl.acm.org/doi/10.1145/2694344.2694380>.
- [34] Computing, Trusted. TCG platform reset attack mitigation specification, 2008. <https://trustedcomputinggroup.org/wp-content/uploads/Platform-Reset-Attack-Mitigation-Specification.pdf>.
- [35] Nuvoton Technology Corporation. NuMicro®Family M2351 Series Datasheet, 2019. https://www.nuvoton.com/export/resource-files/DS_M2351_Series_EN_Rev1.01.pdf.
- [36] Nuvoton Technology Corporation. NuMicro®Family M251/M252 Series Datasheet, 2020. https://www.nuvoton.com/export/resource-files/DS_M251_M252_Series_EN_Rev1.01.pdf.
- [37] Nuvoton Technology Corporation. NuMicro®Family M261/M262/M263 Series Datasheet, 2020. https://www.nuvoton.com/export/resource-files/DS_M261_M262_M263_Series_EN_Rev1.02.pdf.
- [38] Mafalda Cortez, Apurva Dargar, Said Hamdioui, and Geert-Jan 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)*, pages 1–6. IEEE, 2012.
- [39] CPJ. 10 most censored countries, 2020. <https://cpj.org/reports/2019/09/10-most-censored-eritrea-north-korea-turkmenistan-journalist/>.
- [40] Sham M Datta, Vincent J Zimmer, Kushagra V Vaid, William A Stevens, and Amy Lynn Santoni. Processor Cache Memory as RAM for Execution of Boot Code, August 7 2007. US Patent 7,254,676.

- [41] H. Dogan, D. Forte, and M. M. Tehranipoor. Aging analysis for recycled FPGA detection. In *Intl. Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, DFT, pages 171–176, October 2014. doi: 10.1109/DFT.2014.6962099.
- [42] Xintao Duan, Liu Nao, Gou Mengxiao, Dongli Yue, Zimei Xie, Yuanyuan Ma, and Chuan Qin. High-capacity image steganography based on improved fc-densenet. *IEEE Access*, 8: 170174–170182, 2020.
- [43] EET-asia. STMicroelectronics and NXP launch ARM Cortex-M33 Based Chips in Bid to Secure the IoT, 2021. <https://www.nxp.com.cn/docs/en/application-note/AN13037>.<https://www.eetasia.com/18101805-mcus-answer-to-iot-security-worries/>.
- [44] Renesas Electronics. Renesas Synergy Platform Synergy Microcontrollers S1 Series, 2019. <https://www.renesas.com/us/en/document/man/s1ja-microcontroller-group-users-manual>.
- [45] Raspberry Pi Foundation. RPi-3B-Reduced, 2015. https://www.raspberrypi.org/documentation/hardware/raspberrypi/schematics/rpi_SCH_3b_1p2_reduced.pdf.
- [46] RASPBERRY PI FOUNDATION. Raspberry Pi 3 Model B:Single-board computer with wireless LAN and Bluetooth connectivity, 2021. <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>.
- [47] RASPBERRY PI FOUNDATION. Raspberry Pi 4, 2021. <https://www.raspberrypi.org/products/raspberry-pi-4-model-b/>.
- [48] Achiranshu Garg and Tony 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)*, pages 1941–1944. IEEE, 2014.
- [49] Behrad Garmany and Tilo Müller. Prime: Private rsa infrastructure for memory-less encryption. In *Proceedings of the 29th Annual Computer Security Applications Conference, ACSAC*

- '13, page 149158, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450320153. doi: 10.1145/2523649.2523656. URL <https://doi.org/10.1145/2523649.2523656>.
- [50] Le Guan, Jingqiang Lin, Bo Luo, and Jiwu Jing. Copker: Computing with private keys without ram. In *NDSS*, pages 23–26, 2014.
- [51] Le Guan, Jingqiang Lin, Ziqiang Ma, Bo Luo, Luning Xia, and Jiwu Jing. Copker: a cryptographic engine against cold-boot attacks. *IEEE Transactions on Dependable and Secure Computing*, 15(5):742–754, 2016.
- [52] Roberto Guanciale, Hamed Nemati, Christoph Baumann, and Mads Dam. Cache Storage Channels: Alias-driven Attacks and Verified Countermeasures. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 38–55. IEEE, 2016.
- [53] The Guardian. Chinese border guards put secret surveillance app on tourists' phones, 2020. <https://www.theguardian.com/world/2019/jul/02/chinese-border-guards-surveillance-app-tourists-phones>.
- [54] Ujjwal Guin, Xuehui Zhang, Domenic Forte, and Mohammad Tehranipoor. Low-cost On-Chip Structures for Combating Die and IC Recycling. In *Proceedings of the 51st Annual Design Automation Conference, DAC*, pages 87:1–87:6, June 2014. ISBN 978-1-4503-2730-5. doi: 10.1145/2593069.2593157.
- [55] Ujjwal Guin, Wendong Wang, Charles Harper, and Adit D Singh. Detecting recycled socs by exploiting aging induced biases in memory cells. In *2019 IEEE Intl. Symposium on Hardware Oriented Security and Trust (HOST)*, pages 72–80. IEEE, 2019.
- [56] Ujjwal Guin, Wendong Wang, Charles Harper, and Adit D. Singh. Detecting Recycled SoCs by Exploiting Aging Induced Biases in Memory Cells. In *2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 72–80, 2019.

- [57] L. Guo, J. Ni, and Y. Q. Shi. Uniform embedding for efficient JPEG steganography. *IEEE Transactions on Information Forensics and Security*, 9(5):814–825, 2014. doi: 10.1109/TIFS.2014.2312817.
- [58] Peter Gutmann. Data remanence in semiconductor devices. In *USENIX Security Symposium*, pages 39–54, 2001.
- [59] J Alex Halderman, Seth D Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A Calandrino, Ariel J Feldman, Jacob Appelbaum, and Edward W Felten. Lest we remember: cold-boot attacks on encryption keys. *Communications of the ACM*, 52(5):91–98, 2009.
- [60] S. Han, B. S. Kim, and J. Kim. Variation-Aware Aging Analysis in Digital ICs. *Transactions on Very Large Scale Integration Systems*, 21(12):2214–2225, December 2013. ISSN 1063-8210. doi: 10.1109/TVLSI.2012.2228886.
- [61] Helena Handschuh. Hardware-anchored security based on SRAM PUFs, part 1. *IEEE Security & Privacy*, 10(3):80–83, 2012.
- [62] Michael Henson and Stephen Taylor. Beyond full disk encryption: Protection on security-enhanced commodity processors. In *International Conference on Applied Cryptography and Network Security*, pages 307–321. Springer, 2013.
- [63] 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*, 58(9): 1198–1210, September 2009. ISSN 0018-9340. doi: 10.1109/TC.2008.212.
- [64] Daniel E Holcomb, Amir Rahmati, Mastooreh Salajegheh, Wayne P Burleson, and Kevin Fu. Drv-fingerprinting: Using data retention voltage of sram cells for chip identification. In *International Workshop on Radio Frequency Identification: Security and Privacy Issues*, pages 165–179. Springer, 2012.

- [65] Daniel E. Holcomb, Amir Rahmati, Mastooreh Salajegheh, Wayne P. Burleson, and Kevin Fu. DRV-Fingerprinting: Using Data Retention Voltage of SRAM Cells for Chip Identification. In *Proceedings of the 8th Intl. Conference on Radio Frequency Identification: Security and Privacy Issues*, RFIDSec, pages 165–179, 2012.
- [66] Daniel E. Holcomb, Amir Rahmati, Mastooreh Salajegheh, Wayne P. Burleson, and Kevin Fu. DRV-Fingerprinting: Using Data Retention Voltage of SRAM Cells for Chip Identification. In *Proceedings of the 8th Intl. Conference on Radio Frequency Identification: Security and Privacy Issues*, RFIDSec’12, pages 165–179, Berlin, Heidelberg, 2013. Springer-Verlag. ISBN 978-3-642-36139-5. doi: 10.1007/978-3-642-36140-1_12. event-place: Nijmegen, The Netherlands.
- [67] Michael Hutter and Jörn-Marc Schmidt. The Temperature Side Channel and Heating Fault Attacks. In *International Conference on Smart Card Research and Advanced Applications*, pages 219–235. Springer, 2013.
- [68] NewAE Technology Inc., 2021. <https://rtfm.newae.com/Targets/CW308%20UF0/>.
- [69] Texas Instruments. MSP430G2x53 Automotive Mixed-Signal Microcontrollers, 2014. https://www.ti.com/lit/ds/symlink/msp430g2553-q1.pdf?ts=1618423366624&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FMSP430G2553-Q1.
- [70] Texas Instruments. SimplelinkTM ultra-low-power 32-bit arm cortex-m4f mcu with precision adc, 256kb flash and 64kb ram, 2019. https://www.ti.com/lit/ds/symlink/msp432p401r.pdf?ts=1604876136889&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FMSP432P401R.
- [71] Texas Instruments. SimpleLink ultra-low-power 32-bit ARM Cortex-M4F MCU With Precision ADC, 256KB Flash and 64KB RAM, 2019. <https://www.ti.com/document-viewer/MSP432P401R/datasheet/device-overview-slas8261807#SLAS8261807>.

- [72] Intel. 8th and 9th Generation Intel Core Processor Families and Intel Xeon E Processor Families, 2021. <https://www.intel.com/content/dam/www/public/us/en/documents/datasheets/8th-gen-core-family-datasheet-vol-1.pdf#page=118&zoom=100,94,402>.
- [73] Nisha Jacob, Johann Heyszl, Andreas Zankl, Carsten Rolfes, and Georg Sigl. How to Break Secure Boot on FPGA SoCs Through Malicious Hardware. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 425–442. Springer, 2017.
- [74] Shijie Jia, Luning Xia, Bo Chen, and Peng Liu. Deftl: Implementing plausibly deniable encryption in flash translation layer. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2217–2229, 2017.
- [75] K. Kang, H. Kufluoglu, K. Roy, and M. Ashraful Alam. Impact of Negative-Bias Temperature Instability in Nanoscale SRAM Array: Modeling and Analysis. *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(10):1770–1781, October 2007. ISSN 0278-0070. doi: 10.1109/TCAD.2007.896317.
- [76] Naghmeh Karimi, Arun Karthik Kanuparthi, Xueyang Wang, Ozgur Sinanoglu, and Ramesh Karri. Magic: Malicious Aging in Circuits/Cores. *ACM Transactions on Architecture and Code Optimization (TACO)*, 12(1):1–25, 2015.
- [77] Naghmeh Karimi, Arun Karthik Kanuparthi, Xueyang Wang, Ozgur Sinanoglu, and Ramesh Karri. MAGIC: Malicious Aging in Circuits/Cores. *Transaction on Architecture and Code Optimization*, 12(1):5:1–5:25, 2015. ISSN 1544-3566. doi: 10.1145/2724718.
- [78] Naghmeh Karimi, Thorben Moos, and Amir Moradi. Exploring the effect of device aging on static power analysis attacks. *UMBC Faculty Collection*, 2019.
- [79] Sean Keane. Apple AirTags Apparently Hacked by Security Researcher, 2019. <https://www.cnet.com/tech/mobile/apple-airtags-apparently-hacked-by-security-researcher/>.

- [80] Seyab Khan, Nor Zaidi Haron, Said Hamdioui, and Francky Catthoor. NBTI Monitoring and Design for Reliability in Nanoscale Circuits. In *2011 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, pages 68–76. IEEE, 2011.
- [81] Saman Kiamehr, Mohammad Saber Golanbari, and Mehdi B Tahoori. Leveraging aging effect to improve sram-based true random number generators. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 882–885. IEEE, 2017.
- [82] Jeremie S Kim, Minesh Patel, Hasan Hassan, Lois Orosa, and Onur Mutlu. D-range: Using commodity dram devices to generate true random numbers with low latency and high throughput. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 582–595. IEEE, 2019.
- [83] Tony T Kim and Zhi Hui Kong. Impacts of NBTI/PBTI on SRAM V_{min} and Design Techniques for SRAM V_{min} Improvement. In *2011 International SoC Design Conference*, pages 163–166. IEEE, 2011.
- [84] Wonyoung Kim, Meeta S Gupta, Gu-Yeon Wei, and David M Brooks. Enabling On-chip Switching Regulators for Multi-core Processors Using Current Staggering. *Proceedings of the Work. on Architectural Support for Gigascale Integration*, 2007.
- [85] Wonyoung Kim, Meeta S Gupta, Gu-Yeon Wei, and David Brooks. System Level Analysis of Fast, Per-core DVFS Using On-chip Switching Regulators. In *2008 IEEE 14th International Symposium on High Performance Computer Architecture*, pages 123–134. IEEE, 2008.
- [86] Robert R. King. North koreans want external information, but Kim Jong-Un seeks to limit access, May 2019. <https://www.csis.org/analysis/north-koreans-want-external-information-kim-jong-un-seeks-limit-access>.
- [87] Florian Klemme and Hussam Amrouch. Machine learning for circuit aging estimation under workload dependency. In *2021 IEEE International Test Conference (ITC)*, pages 37–46. IEEE, 2021.

- [88] G Knight. Encrypt data using veracrypt. 2017.
- [89] Paul Kocher, Jann Horn, Anders Fogh, , Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. In *40th IEEE Symposium on Security and Privacy (S&P'19)*, 2019.
- [90] Sanjay V Kumar, Chris H Kim, and Sachin S Sapatnekar. An Analytical Model for Negative Bias Temperature Instability. In *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*, pages 493–496, 2006.
- [91] Edlyn V. Levine. Hardware is a cybersecurity risk. Here's what we need to know, 2019. URL <https://www.weforum.org/agenda/2019/12/our-hardware-is-under-cyberattack-heres-how-to-make-it-safe/>. Accessed: 2024.
- [92] Arm Limited. Arm cortex-a72 mpcore processor technical reference manual, 2016. <https://developer.arm.com/documentation/100095/0003/>.
- [93] ARM limited. Arm Cortex-A78 Core Technical Reference Manual, 2021. <https://developer.arm.com/documentation/102160/latest>.
- [94] Simon Lindenlauf, Hans Höfken, and Marko Schuba. Cold boot attacks on ddr2 and ddr3 sdram. In *2015 10th International Conference on Availability, Reliability and Security*, pages 287–292. IEEE, 2015.
- [95] Moritz Lipp, Daniel Gruss, Raphael Spreitzer, Clémentine Maurice, and Stefan Mangard. Armageddon: Cache Attacks on Mobile Devices. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 549–564, 2016.
- [96] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg.

- Meltdown: Reading kernel memory from user space. In *27th USENIX Security Symposium (USENIX Security 18)*, 2018.
- [97] T. Liu, C. C. Chen, J. Wu, and L. Milor. SRAM stability analysis for different cache configurations due to Bias Temperature Instability and Hot Carrier Injection. In *Intl. Conference on Computer Design, ICCD*, pages 225–232, October 2016. doi: 10.1109/ICCD.2016.7753284.
- [98] Taizhi Liu, Chang-Chih Chen, Jiadong Wu, and Linda Milor. Sram stability analysis for different cache configurations due to bias temperature instability and hot carrier injection. In *2016 IEEE 34th International Conference on Computer Design (ICCD)*, pages 225–232. IEEE, 2016.
- [99] Xing Liu, Cheng Qian, William Grant Hatcher, Hansong Xu, Weixian Liao, and Wei Yu. Secure internet of things (iot)-based smart-world critical infrastructures: Survey, case study and research opportunities. *IEEE Access*, 7:79523–79544, 2019.
- [100] R. Maes and V. van der Leest. Countering the effects of silicon aging on SRAM PUFs. In *Intl. Symposium on Hardware-Oriented Security and Trust, HOST*, pages 148–153, May 2014. doi: 10.1109/HST.2014.6855586.
- [101] 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)*, pages 148–153, 2014. doi: 10.1109/HST.2014.6855586.
- [102] Heather Mahalik. 5 questions that will unlock more data with checkm8, February 2020. <https://www.cellebrite.com/en/blog/iphone-extractions-5-questions-to-unlock-more-data/>.
- [103] Jubayer Mahmud and Matthew Hicks. Invisible bits: hiding secret messages in srams analog domain. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 1086–1098, 2022.

- [104] Jubayer Mahmod and Matthew 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*, pages 1043–1055, 2022.
- [105] Jubayer Mahmod and Matthew Hicks. Untrustzone: Systematic accelerated aging to expose on-chip secrets. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 69–69. IEEE Computer Society, 2024.
- [106] A. Maiti, L. McDougall, and P. Schaumont. The Impact of Aging on an FPGA-Based Physical Unclonable Function. In *Intl. Conference on Field Programmable Logic and Applications, FPL*, pages 151–156, September 2011. doi: 10.1109/FPL.2011.35. star.
- [107] Abhranil Maiti, Logan McDougall, and Patrick Schaumont. The impact of aging on an fpga-based physical unclonable function. In *2011 21st International Conference on Field Programmable Logic and Applications*, pages 151–156. IEEE, 2011.
- [108] Abhranil Maiti, Vikash Gunreddy, and Patrick Schaumont. A Systematic Method to Evaluate and Compare the Performance of Physical Unclonable Functions. In *Embedded systems design with FPGAs*, pages 245–267. Springer, 2013.
- [109] Cedric Marchand, Alain Aubert, Lilian Bossuet, et al. On the Security Evaluation of the ARM TrustZone Extension in a Heterogeneous SoC. In *2017 30th IEEE International System-on-Chip Conference (SOCC)*, pages 108–113. IEEE, 2017.
- [110] Bernard Marr. Smart Dust is Coming. Are You Ready. Accessed August 2023, 30, 2018. <https://www.forbes.com/sites/bernardmarr/2018/09/16/smart-dust-is-coming-are-you-ready/?sh=42c415765e41>.
- [111] Andrew D McDonald and Markus G Kuhn. StegFS: A steganographic file system for linux. In *International Workshop on Information Hiding*, pages 463–477. Springer, 1999.

- [112] J. McMahan, W. Cui, L. Xia, J. Heckey, F. T. Chong, and T. Sherwood. Challenging On-chip SRAM Security with Boot-state Statistics. In *2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 101–105, 2017. doi: 10.1109/HST.2017.7951806.
- [113] Joseph W McPherson. Reliability challenges for 45nm and beyond. In *2006 43rd ACM/IEEE Design Automation Conference*, pages 176–181, 2006.
- [114] Microchip Technology Inc. SAM L10/L11 Family: Ultra Low-Power, 32-bit Cortex-M23 MCUs with TrustZone, Crypto, and Enhanced PTC, 2020.
- [115] Microchip Technology Inc. World-Class, Award-Winning SAM L10 and SAM L11 Microcontroller Family, 2021. <https://www.microchip.com/en-us/products/microcontrollers-and-microprocessors/32-bit-mcus/sam-32-bit-mcus/sam-l/sam-l10-l11>.
- [116] Microsoft. Overview of BitLocker Device Encryption in Windows 10, 2019. <https://docs.microsoft.com/en-us/windows/security/information-protection/bitlocker/bitlocker-device-encryption-overview-windows-10>.
- [117] Praveen Mosalikanti, Chris Mozak, and Nasser Kurd. High performance ddr architecture in intel® core processors using 32nm cmos high-k metal-gate process. In *Proceedings of 2011 International Symposium on VLSI Design, Automation and Test*, pages 1–4. IEEE, 2011.
- [118] J Mull. How a syrian refugee risked his life to bear witness to atrocities. *Toronto Star Online*, posted, 2012.
- [119] Tilo Müller and Michael Spreitzenbarth. Frost: Forensic recovery of scrambled telephones. In *Proceedings of the 11th International Conference on Applied Cryptography and Network Security, ACNS’13*, page 373388, Berlin, Heidelberg, 2013. Springer-Verlag. ISBN 9783642389795. doi: 10.1007/978-3-642-38980-1_23. URL https://doi.org/10.1007/978-3-642-38980-1_23.

- [120] Tilo Müller, Felix C. Freiling, and Andreas Dewald. Tresor runs encryption securely outside ram. In *Proceedings of the 20th USENIX Conference on Security, SEC'11*, page 17, USA, 2011. USENIX Association.
- [121] NXP Semiconductor. <https://www.nxp.com/docs/en/application-note/AN13037.pdf>. Accessed: 2023-7-15.
- [122] NXP Semiconductors. UM11126:LPC55S6x/LPC55S2x/LPC552x User manual, 2019.
- [123] NXP Semiconductors. i.MX535: Multimedia Applications Processors - HD Video, High-Performance, Low-Power, ARM Cortex®-A8 Core, 2021. <https://www.nxp.com/products/processors-and-microcontrollers/arm-processors/i-mx-applications-processors/i-mx-mature-processors/multimedia-applications-processors-hd-video-high-performance-low-power-arm-cortex-a8-core-i.MX535>.
- [124] NXP Semiconductors. LPC55S6x: High Efficiency Arm® Cortex®-M33-Based Microcontroller Family, 2021. <https://www.nxp.com/products/processors-and-microcontrollers/arm-microcontrollers/general-purpose-mcus/lpc5500-cortex-m33/high-efficiency-arm-cortex-m33-based-microcontroller-family:LPC55S6x>.
- [125] Lois Orosa, Yaohua Wang, Mohammad Sadrosadati, Jeremie S. Kim, Minesh Patel, Ivan Puddu, Haocong Luo, Kaveh Razavi, Juan Gómez-Luna, Hasan Hassan, Nika Mansouri-Ghiasi, Saugata Ghose, and Onur Mutlu. Codic: A low-cost substrate for enabling custom in-dram functionalities and optimizations. In *Proceedings of the 48th Annual International Symposium on Computer Architecture, ISCA '21*, page 484497. IEEE Press, 2021. ISBN 9781450390866. doi: 10.1109/ISCA52012.2021.00045. URL <https://doi.org/10.1109/ISCA52012.2021.00045>.
- [126] Jürgen Pabel. FrozenCache Mitigating Cold-boot Attacks for Full-Disk-Encryption Software. In *27th Chaos Communication Congress*, 2010.

- [127] Timothy M Peters, Mark A Gondree, and Zachary NJ Peterson. DEFY: A deniable, encrypted file system for log-structured storage. 2015.
- [128] Raspberry Pi. Raspberry pi 4 model b specifications. URL <https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-reduced-schematics.pdf>.
- [129] Hulfang Qin, Yu Cao, Dejan Markovic, Andrei Vladimirescu, and Jan Rabaey. SRAM leakage suppression by minimizing standby supply voltage. In *International Symposium on Signals, Circuits and Systems. Proceedings, SCS 2003.(Cat. No. 03EX720)*, pages 55–60. IEEE, 2004.
- [130] Pengfei Qiu, Dongsheng Wang, Yongqiang Lyu, and Gang Qu. VoltJockey: Breaching Trust-Zone by Software-Controlled Voltage Manipulation Over Multi-Core Frequencies. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 195–209, 2019.
- [131] M Tauhidur Rahman, Domenic Forte, Xiaoxiao Wang, and Mark Tehranipoor. Enhancing Noise Sensitivity of Embedded SRAMs for Robust True Random Number Generation in SoCs. In *2016 IEEE Asian Hardware-Oriented Security and Trust (AsianHOST)*, pages 1–6. IEEE, 2016.
- [132] Amir Rahmati, Mastooreh Salajegheh, Dan Holcomb, Jacob Sorber, Wayne P Burleson, and Kevin Fu. {TARDIS}: Time and remanence decay in {SRAM} to implement secure protocols on embedded devices without clocks. In *21st {USENIX} Security Symposium ({USENIX} Security 12)*, pages 221–236, 2012.
- [133] Raspberry Pi Foundation. Boot Sequence, 2021. <https://www.raspberrypi.org/documentation/hardware/raspberrypi/bootmodes/bootflow.md>.
- [134] Raspberry Pi Foundation. Raspberry Pi 3 Model B (v1.2), 2021. <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>.

- [135] Raspberry Pi Ltd. Raspberry pi 4 model B specifications –. <https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/>. Accessed: 2023-1-10.
- [136] Dominic Rath. Open On-Chip Debugger: Free and Open On-Chip Debugging, In-System Programming and Boundary-Scan Testing, 2021. <http://openocd.org/about/>.
- [137] Kaveh Razavi, Ben Gras, Erik Bosman, Bart Preneel, Cristiano Giuffrida, and Herbert Bos. Flip Feng Shui: Hammering a Needle in the Software Stack. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 1–18, 2016.
- [138] RICHTEK. 2.7MHz 3A Step-Down Converter with I²C Interface, 2013. https://www.richtek.com/assets/product_file/RT8088A/DS8088A-00.pdf.
- [139] A. Roelke and M. R. Stan. Attacking an SRAM-Based PUF through Wearout. In *IEEE Computer Society Annual Symposium on VLSI, ISVLSI*, pages 206–211, July 2016. doi: 10.1109/ISVLSI.2016.68.
- [140] Alec Roelke and Mircea R Stan. Attacking an sram-based puf through wearout. In *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 206–211. IEEE, 2016.
- [141] Alec Roelke and Mircea R. Stan. Controlling the reliability of sram pufs with directed nbtI aging and recovery. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 26(10):2016–2026, 2018. doi: 10.1109/TVLSI.2018.2836154.
- [142] Alec Roelke and Mircea R. Stan. Controlling the Reliability of SRAM PUFs With Directed NBTI Aging and Recovery. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 26(10):2016–2026, 2018.
- [143] Ulrich Ruhrmair. SoK: Towards secret-free security. In *Proceedings of the 4th ACM Workshop on Attacks and Solutions in Hardware Security*, pages 5–19, 2020.

- [144] Keegan Ryan. Hardware-backed heist: extracting ecDSA keys from qualcomm's trustzone. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 181–194, 2019.
- [145] Sachin S Sapatnekar. What happens when circuits grow old: Aging issues in CMOS design. In *2013 Intl. Symposium on VLSI Technology, Systems and Application (VLSI-TSA)*, pages 1–2, 2013.
- [146] Tobias Schmidbauer and Steffen Wendzel. Covert storage caches using the NTP protocol. In *Proceedings of the 15th International Conference on Availability, Reliability and Security*, pages 1–10, 2020.
- [147] Adel S Sedra, Dean Emeritus Adel S Sedra, Kenneth Carless Smith, and Kenneth C Smith. *Microelectronic circuits*. New York: Oxford University Press, 1998.
- [148] Nader Sehatbakhsh, Hope Hong, Benjamin Lazar, Barry Johnson-Smith, Oghuzhan Yilmaz, Monjur Alam, Alireza Nazari, Alenka Zajic, and Milos Prvulovic. Syndrome: Spectral analysis for anomaly detection on medical iot and embedded devicesexperimental demonstration. In *Hardware Demo at IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2017.
- [149] NXP Semiconductors. LPC55Sxx debug authentication, 2021. <https://www.nxp.com.cn/docs/en/application-note/AN13037.pdf>.
- [150] Deepashree Sengupta and Sachin S Sapatnekar. Estimating circuit aging due to bti and hci using ring-oscillator-based sensors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(10):1688–1701, 2017.
- [151] Hoseok Seol, Minhye Kim, Taesoo Kim, Yongdae Kim, and Lee-Sup Kim. Amnesiac dram: A proactive defense mechanism against cold boot attacks. *IEEE Transactions on Computers*, 70(4):539–551, 2021. doi: 10.1109/TC.2019.2946365.

- [152] Mehdi Sharifzadeh, Mohammed Aloraini, and Dan Schonfeld. Adaptive batch size image merging steganography and quantized gaussian image steganography. *IEEE Transactions on Information Forensics and Security*, 15:867–879, 2019.
- [153] Shikha Sharma and Devendra Somwanshi. A DWT based attack resistant video steganography. In *Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies*, pages 1–5, 2016.
- [154] Jeonghee Shin, Victor Zyuban, Pradip Bose, and Timothy M Pinkston. A proactive wearout recovery approach for exploiting microarchitectural redundancy to extend cache sram lifetime. *ACM SIGARCH Computer Architecture News*, 36(3):353–362, 2008.
- [155] SiFive, Inc. SiFive U54-MC manual, 2019. https://simfive.cdn.prismic.io/sifive%2Fdc4980ff-17db-448b-b521-4c7ab26b7488_sifive+u54-mc+manual+v19.08.pdf.
- [156] Silicon Labs. EFM32 Wonder Gecko Family EFM32WG Data Sheet, 2012. <https://www.silabs.com/documents/public/data-sheets/efm32wg-datasheet.pdf>.
- [157] Patrick Simmons. Security through amnesia: a software-based solution to the cold boot attack on disk encryption. In *Proceedings of the 27th Annual Computer Security Applications Conference*, pages 73–82, 2011.
- [158] Adam Skillen and Mohammad Mannan. On implementing deniable storage encryption for mobile devices. 2013.
- [159] Adam Skillen and Mohammad Mannan. Mobiflage: Deniable storage encryption for mobile devices. *IEEE Transactions on Dependable and Secure Computing*, 11(3):224–237, 2013.
- [160] Sergei Skorobogatov. Low temperature data remanence in static ram. Technical report, University of Cambridge, Computer Laboratory, 2002.
- [161] Sergei Petrovich Skorobogatov. Semi-invasive Attacks: a New Approach to Hardware Security Analysis. 2005.

- [162] Avinash Srinivasan, Jie Wu, Panneer Santhalingam, and Jeffrey Zamanski. Deaddrop-in-a-flash: Information hiding at ssd nand flash memory physical layer. *SECURWARE 2014*, page 79, 2014.
- [163] STMicroelectronics. Ultra-low-power Arm[®] Cortex-M33 32-bit MCU+TrustZone[®]+FPU, 165DMIPS, up to 512KB Flash, 256KB SRAM, SMPS, AES+PKA, 2020. <https://www.st.com/resource/en/datasheet/stm32l562ce.pdf>.
- [164] Adrian Tang, Simha Sethumadhavan, and Salvatore Stolfo. {CLKSCREW}: Exposing the Perils of Security-oblivious Energy Management. In *26th {USENIX} Security Symposium ({USENIX} Security 17)*, pages 1057–1074, 2017.
- [165] Fatemeh Tehranipoor, Wei Yan, and John A Chandy. Robust hardware true random number generators using dram remanence effects. In *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 79–84. IEEE, 2016.
- [166] TestEquity. Model 123H Temperature/Humidity Chamber, 2021. <https://www.testequity.com/category/Environmental-Chambers-Ovens/Temperature-Humidity-Chambers/TestEquity-123H-Temperature-Humidity-Chamber-North-America-Version-17267-1>.
- [167] Muller Tilo, Spreitzenbarth Michael, and Felix C Freiling. Frost: Forensic Recovery of Scrambled Telephones. In *International Conference on Applied Cryptography and Network Security*, 2014.
- [168] Timothy Trippel, Kang G Shin, Alex Chernyakhovsky, Garret Kelly, Dominic Rizzo, and Matthew Hicks. Fuzzing hardware like software. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 3237–3254, 2022.
- [169] Tim Tuan, Tom Strader, and Steve Trimberger. Analysis of Data Remanence in a 90nm FPGA. In *2007 IEEE Custom Integrated Circuits Conference*, pages 93–96. IEEE, 2007.

- [170] B. Tudor, Joddy Wang, W. Liu, and Hany Elhak. Mos device aging analysis with hspice and customsim. 2011.
- [171] United States of America Central Intelligence Agency. TRIGON: Spies passing in the night, 2016. <https://www.cia.gov/news-information/featured-story-archive/2016-featured-story-archive/trigon-spies-passing-in-the-night.html>.
- [172] United States of America Customs and Border Protection Agency. CBP statement on border search of electronic devices, April 2017. <https://www.cbp.gov/newsroom/national-media-release/cbp-releases-statistics-electronic-device-searches-0#wcm-survey-target-id>.
- [173] United States of America Customs and Border Protection Agency. CBP statement on border search of electronic devices, October 2019. <https://www.cbp.gov/newsroom/speeches-and-statements/cbp-statement-border-search-electronic-devices>.
- [174] Elena Ioana Vatajelu, Giorgio Di Natale, and Paolo Prinetto. Towards a Highly Reliable SRAM-based PUFs. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 273–276. IEEE, 2016.
- [175] K. Wang, H. Zhao, and H. Wang. Video steganalysis against motion vector-based steganography by adding or subtracting one motion vector value. *IEEE Transactions on Information Forensics and Security*, 9(5):741–751, 2014. doi: 10.1109/TIFS.2014.2308633.
- [176] Y. Wang, W. Yu, S. Wu, G. Malysa, G. E. Suh, and E. C. Kan. Flash Memory for Ubiquitous Hardware Security Functions: True Random Number Generation and Device Fingerprints. In *IEEE Symposium on Security and Privacy*, Oakland, pages 33–47, May 2012.
- [177] Y. Wang, W. Yu, S. Q. Xu, E. Kan, and G. E. Suh. Hiding information in flash memory. In *2013 IEEE Symposium on Security and Privacy*, pages 271–285, 2013. doi: 10.1109/SP.2013.26.

- [178] Dipti Watni and Sonal Chawla. A comparative evaluation of jpeg steganography. In *2019 5th International Conference on Signal Processing, Computing and Control (ISPCC)*, pages 36–40. IEEE, 2019.
- [179] Mark Weiser, Rich Gold, and John Seely Brown. The Origins of Ubiquitous Computing Research at PARC in the Late 1980s. *IBM systems journal*, 38(4):693–696, 1999.
- [180] Steffen Wendzel, Benjamin Kahler, and Thomas Rist. Covert channels and their prevention in building automation protocols: A prototype exemplified using BACnet. In *2012 IEEE International Conference on Green Computing and Communications*, pages 731–736. IEEE, 2012.
- [181] Steffen Wendzel, Wojciech Mazurczyk, and Georg Haas. Dont you touch my nuts: Information hiding in cyber physical systems. In *2017 IEEE Security and Privacy Workshops (SPW)*, pages 29–34. IEEE, 2017.
- [182] Harrison Williams, Alexander Lind, Kishankumar Parikh, and Matthew Hicks. Silicon dating. *arXiv preprint arXiv:2009.04002*, 2020.
- [183] Gilson I Wirth, Roberto da Silva, and Ben Kaczer. Statistical Model for MOSFET Bias Temperature Instability Component Due to Charge Trapping. *IEEE Transactions on Electron Devices*, 58(8):2743–2751, 2011.
- [184] Inc Xilinx. Zynq-7000 SoC and 7 Series Devices Memory Interface Solutions, 2018. https://www.xilinx.com/support/documentation/ip_documentation/mig_7series/v4_2/ds176_7Series_MIS.pdf.
- [185] Yunzhao Yang, Yuntao Wang, Xiaowei Yi, Xianfeng Zhao, and Yi Ma. Defining joint embedding distortion for adaptive MP3 steganography. In *Proceedings of the ACM Workshop on Information Hiding and Multimedia Security*, pages 14–24, 2019.

- [186] Salessawi Ferede Yitbarek, Misiker Tadesse Aga, Reetuparna Das, and Todd Austin. Cold Boot Attacks are Still Hot: Security Analysis of Memory Scramblers in Modern Processors. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 313–324, Austin, TX, USA, February 2017. IEEE. ISBN 978-1-5090-4985-1. doi: 10.1109/HPCA.2017.10. URL <http://ieeexplore.ieee.org/document/7920835/>.
- [187] Sebastian Zander, Grenville Armitage, and Philip Branch. A survey of covert channels and countermeasures in computer network protocols. *IEEE Communications Surveys & Tutorials*, 9(3):44–57, 2007.
- [188] Shaza Zeitouni, Yossef Oren, Christian Wachsmann, Patrick Koeberl, and Ahmad-Reza Sadeghi. Remanence Decay Side-channel: The PUF Case. *IEEE Transactions on Information Forensics and Security*, 11(6):1106–1116, 2015.
- [189] Fengchao Zhang, Shuo Yang, Jim Plusquellic, and Swarup Bhunia. Current Based PUF Exploiting Random Variations in SRAM Cells. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 277–280. IEEE, 2016.
- [190] Ning Zhang, Kun Sun, Wenjing Lou, and Y Thomas Hou. Case: Cache-assisted secure execution on arm processors. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 72–90. IEEE, 2016.
- [191] Ning Zhang, Kun Sun, Deborah Shands, Wenjing Lou, and Y Thomas Hou. TruSpy: Cache Side-Channel Information Leakage from the Secure World on ARM Devices. *IACR Cryptol. ePrint Arch.*, 2016:980, 2016.
- [192] Rui Zheng, Jyothi Velamala, Vijay Reddy, Varsha Balakrishnan, Evelyn Mintarno, Subhasish Mitra, Srikanth Krishnan, and Yu Cao. Circuit aging prediction for low-power operation. In *2009 IEEE Custom Integrated Circuits Conference*, pages 427–430, 2009. doi: 10.1109/CICC.2009.5280814.

- [193] Aviad Zuck, Yue Li, Jehoshua Bruck, Donald E. Porter, and Dan Tsafir. Stash in a flash. In *16th USENIX Conference on File and Storage Technologies (FAST 18)*, pages 169–188, Oakland, CA, February 2018. USENIX Association. ISBN 978-1-931971-42-3. URL <https://www.usenix.org/conference/fast18/presentation/zuck>.