A Survey on Hardware Security Techniques Targeting Low-Power SoC Designs

Alan Ehret
Adaptive and Secure Computing Systems (ASCS) Laboratory
Department of Electrical and Computer Engineering
Boston University
ehretaj@bu.edu

Karen Gettings
MIT Lincoln Laboratory
Lexingon, United States of America
karen.gettings@ll.mit.edu

Bruce R. Jordan Jr.
MIT Lincoln Laboratory
Lexingon, United States of America
bruce.jordan@ll.mit.edu

Michel A. Kinsy
Adaptive and Secure Computing Systems (ASCS) Laboratory
Department of Electrical and Computer Engineering
Boston University
mkinsy@bu.edu

Abstract—**In this work, we survey hardware-based security techniques applicable to low-power system-on-chip designs. Techniques related to a system's processing elements, volatile main memory and caches, non-volatile memory and on-chip interconnects are examined. Threat models for each subsystem and technique are considered. Performance overheads and other trade-offs for each technique are discussed. Defenses with similar threat models are compared.**

Keywords—Hardware Security, System-on-Chip, Secure Enclave, PUF, Network-on-chip, Oblivious RAM

I. Introduction

Modern computing systems support more computation on smaller devices than ever before. The integration of memory, CPU cores, and hardware accelerators on a single die has led to System-on-Chip (SoC) designs suitable for a plethora of embedded systems and mobile devices. For example, interconnected sensor devices, i.e. the internet-of-things (IoT), are often engineered to be deployed in large numbers to cover and relay user commands over a large area. These microsystems may be located outside a secure perimeter (e.g., outdoor public places) and may require an additional level of security to prevent attacks that can compromise their operation. However, securing an SoC against particular threats can carry a high overhead, which is sometimes difficult to fulfill in power-constrained embedded and mobile systems. Additionally, threats faced by SoCs may not be possible to mitigate with software solutions alone. Attacks such as power and timing side-channels or memory probing will require hardware based defenses. To examine which hardware defenses are suitable for low-power SoC designs, we survey a variety of common hardware based security techniques.

The scope of this work covers hardware-based security techniques related to System-on-Chip (1) processing elements including general purpose CPU cores and dedicated hardware accelerators, (2) volatile memory, such as caches and main memory, (3) non-volatile memories and (4) on-chip interconnects such as, Networks-on-Chip (NoC). We examine which threat models are covered by each technique, how the threats are mitigated, and the reported overheads associated with each technique. Additionally, we discuss how each technique may apply to low-power SoC systems. A comparison of techniques and threats each mitigates is presented.

DISTRIBUTION STATEMENT A. Approved for public release. Distribution is unlimited. This material is based upon work supported by the Under Secretary of Defense for Research and Engineering under Air Force Contract No. FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Under Secretary of Defense for Research and Engineering.

II. Processing Element Techniques

In this section, we examine hardware-based techniques to defend processing elements in a SoC against a variety of threat models. In this context, any non-memory component (we consider *caches* to be memory for the purposes of this survey) connected to a SoC interconnect (such as a bus or NoC) can be considered a processing element. Common processing elements include general purpose CPU cores or dedicated hardware accelerators for tasks such as encryption. System-on-Chip processing elements face a variety of threat models, including malicious software running on trusted processing elements and outside attackers attempting to reveal secret information through power or timing side-channels.

A. Secure Enclaves

Secure enclaves defend against a variety of threats with physical isolation. Physical isolation protects secret data and computation from side-channel attacks as well as direct, unauthorized access. Hardware resources are dedicated to security critical functions, such as encryption or authentication. Low-power SoCs can utilize secure enclaves for tasks that demand the highest level of security. An enclave can operate in a low-power state when it is not in use. Several commercial secure enclave implementations exist.

Apple's Secure Enclave Processor (SEP) [1] is a co-processor that utilizes memory encryption and hardware-based random number generation to carry out cryptographic functions for a main Application Processor (AP). SEP creates a logical wall between untrusted software and sensitive security functions so that untrusted software cannot gain access to sensitive data such as fingerprints and keys. The basic architectural design of SEP is the separation of computation between the AP and SEP processors. In addition to the hardware random number generator, SEP also contains an isolated boot ROM and crypto engine. Despite this aggressive separation, SEP is still a 32-bit processor that coordinates with the AP to share external memory. During its boot process, SEP will wait for AP to configure a region of memory. Communication between AP and SEP is achieved through an interrupt-driven secure mailbox. All data originating from the SoC must go through the secure mailbox to be used by the SEP. Once SEP has initialized secure memory regions, its isolation protects it from software-based attacks. Furthermore, after initialization, applications that wish to interact with the encrypted data guarded by SEP must use a bootstrap server that can enforce access and privilege rules for different functionalities, such as a secure key generation service.

The ARM TrustZone technology [2] is a single core secure processor technology that uses a security approach similar to Apple's Secure Enclave Processor. ARM TrustZone uses separation based on the concept of least privilege; software or hardware should only have access to the compute resources that it needs and nothing more. To implement this secure model, TrustZone creates two logical zones: secure world and non-secure world; the secure world houses the security subsystem, while the non-secure world contains everything else. This allows a chain of trust to be established. Separation of zones is based on secure and non-secure memory partitions.

Intel's Trusted Execution Technology (TXT) [3] is a hardware-based technology to examine the authenticity of the operating system and its running environment. It relies on the Trusted Platform Module (TPM) to provide functionalities such as secure storage. The purpose of the TXT is to provide a trusted mechanism to load and execute system software, e.g., Operating System kernel or Virtualization Machine Monitor (VMM), even on machines with malicious software and malware.

Secure enclaves offer strong protection, but generally limit performance of security critical functions when compared to the performance of their system's main processor. For example, enclaves based on ARM TrustZone that are external to a SoC (such as a smart card or a phone's SIM card) are generally limited to a clock frequency of 5-20MHz [3]. Such a clock frequency is often not suitable for high-performance systems with clock frequencies of several gigahertz. However, low-power SoC systems (such as IoT devices) generally have lower clock frequencies. The lower performance associated with low-power SoC systems means secure enclaves can reasonably be included in low-power SoC designs without drastically restricting SoC performance or violating power/area budgets.

B. *Execution Obfuscation*

The execution of security critical algorithms, such as encryption, can be undermined by the leakage of information through side-channels. Previous works have demonstrated attacks related to timing, power, electro-magnetic (EM), and fault-based side-channel attacks [4] [5] [6]. The threat model for side-channel attacks varies slightly, depending on the side-channel exploited. Attacks that exploit power or EM side-channels often require direct physical control of a device. Meanwhile, timing attacks are not dependent on the same level of direct access or physical control necessary for power and EM based side-channel attacks, but do require some level of remote control over a system. For example, the Spectre and Meltdown class of attacks execute entirely in software and can be exploited without physical access to a system [7] [8].

Execution obfuscation-based techniques attempt to mitigate side-channels without the overhead of the physical isolation used in secure enclaves. One example, a co-processor named Ascend [9], operates on encrypted user inputs with trusted or untrusted programs in a semi-honest server. In the threat model used by the authors, a semi-honest server will correctly execute a given program with the given encrypted user inputs but may also attempt to leak information about the inputs by running other programs with them. The authors describe this semi-honest model as "honest but curious". Figure 1 shows a high-level view of Ascend's computation process.

The Ascend processor uses public/private key pairs to share symmetric keys used to decrypt user inputs and program binaries, preventing the server from directly observing their contents. Power and I/O side-channels are obfuscated by activating major architectural components (cache, memory interface, register file, etc.) for each instruction, whether or not the component is needed. Oblivious RAM, covered in Section III-A, is used to obscure the timing, I/O, and power side-channels related to off-chip memory. In order to mask timing information related to program runtime, Ascend receives a time and power budget from the user. The encrypted program state is returned only after Ascend exhausts the given time and power budget. The program execution may be incomplete if the time and power budget is not large enough. Otherwise, the program results are returned in the program state. Keeping the results encrypted prevents the semi-honest server from learning anything about the plain text of the user input or final result of the program. The semi-honest server only has an estimate of how long a user's operation took to execute. Previous work has shown this to be the smallest possible amount of information to leak about a program execution [10]. The authors evaluate Ascend with several SPEC06int benchmarks and report an average of 13.5x slowdown compared to an equivalent non-obfuscated architecture.

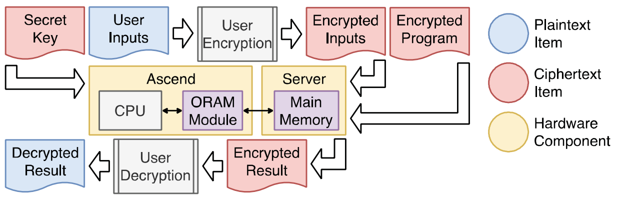

Fig. 1. The flow of Ascend's secret computation process.

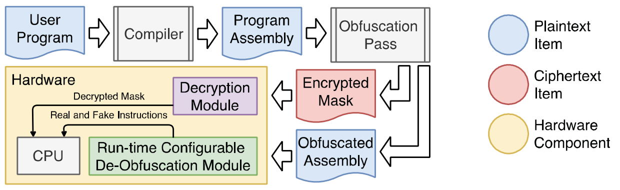

Fig. 2. The hardware/software compiliation and execution flow of Sphinx.

A different architecture, Sphinx [11], supports obfuscation with a hardware-software co-design approach. Sphinx prevents attackers from leaking side-channel information with support for binary obfuscation and a runtime-reconfigurable level of execution flow obfuscation. To obfuscate execution, the compilation flow used by Sphinx inserts random instructions in the binary and provides an encrypted mask that reveals which instructions are real. Reconfiguration allows users to trade-off obfuscation and performance by executing, but not committing, a user defined level of obfuscation instructions in the binary. Multiple compilation runs with the same source will produce a unique binary for each run, preventing attackers from applying knowledge leaked from one deployment to another. Figure 2 outlines the hardware/software flow used by Sphinx. Sphinx provides a weaker but more flexible obfuscation than Ascend, i.e. Ascend is guaranteed to obfuscate most side-channels at all times, while Sphinx will only obfuscate side-channels with a sufficiently high number of executed obfuscation instructions.

However, Sphinx allows users to determine an appropriate obfuscation level (and therefore the difficulty in exploiting side-channels) based on their requirements and maintain much of the performance of un-obfuscated execution. Additionally, the hardware-software co-design approach and customized binaries used by Sphinx provide obfuscation unique to each deployment. Knowledge gained by an attacker about one Sphinx deployment cannot be applied to another deployment.

C. *Physical Unclonable Functions*

A Physical Unclonable Function (PUF) creates a unique identification for each implementation of the same design. The uniqueness of most PUFs results from distinct physical properties of the implementation that cannot be recreated with existing manufacturing techniques. The unique outputs of PUFs can be used for authentication or secret key storage. A PUF's precise output is only retrievable with an input and the physical characteristics of the PUF itself. Although the PUF concept has been known since 1983 [12] [13], the term PUF only came into existence in 2002 [14]. PUF-based technology is a promising technique for silicon device fingerprinting [15] [16]. Researchers are studying and developing different types of PUFs and their implementations.

There are three main classes of physical unclonable functions (PUFs), namely, strong PUFs [14] [17] [18], controlled PUFs [19] [20], and weak PUFs or physically obfuscated keys (POKs) [21, 22]. Each class has its own application target domain and security features. A PUF can be categorized as an explicitly-introduced randomness PUF or intrinsic randomness PUF, based on how the randomness is introduced in [23]. For explicitly-introduced randomness PUFs, optical PUFs (non-electrical) and coating PUFs are the two main sub-classes [24] [25]. Intrinsic randomness PUFs are used more often, because they can be included in a design without modifications to the manufacturing process [26].

A secure PUF must be unpredictable, unclonable, and tamper detectable. Unfortunately, due to implementation challenges or flaws, many PUFs have succumbed to some sort of attack [27][28][29]. Many attacks try to acquire information on the PUF inputs and outputs, known as challenge-response pairs (CRPs). For example, in 2013 several researchers showed that it is possible to clone a SRAM weak PUF by reading the SRAM memories out through either standard on-chip channels or laser stimulation [30] [31]. Strong PUFs allow anyone to hold a large subset of the CRPs; thus, they are vulnerable to modeling attack. Researchers, e.g., [32], [29] [33], have used machine learning techniques to model Arbiter and Ring Oscillator PUFs and have had good success in predicting the PUF's unknown CRPs. Researchers have also studied the information leakage from the PUF's public helper data [34]. In recent works, there have been attempts to combine both modeling and side-channel attacks to improve an attack's effectiveness [35]. This type of attack has successfully modeled some strong and secure PUFs such as XOR Arbiter PUFs and Lightweight Secure PUFs, achieving a 99% successful prediction rate. Besides passive learning attacks, PUFs can also be tampered with physically to alter the response behavior permanently and noticeably.

Despite these shortcomings, PUFs allow authentication or secret storage to be rooted in hardware. Many of the PUF attacks described above are more difficult to exploit (requiring invasive attacks on the hardware itself) than attacks on alternative software-based authentication and key storage schemes. Additionally, the simplicity of PUFs makes their area and power overheads minimal. For these reasons, PUFs provide a suitable option for low-power SoC systems in need of secret key storage or device authentication schemes.

III. Volatile Memory System Techniques

Next, we examine hardware-based security techniques focused on volatile memory systems on SoCs. Volatile memory, such as a cache or main memory, will frequently store secret information, such as encryption keys or decrypted data. Storing such sensitive data makes non-volatile memories valuable targets for attackers. The scope of this paper is limited to cache and on- or off-chip main memory systems. A variety of previous works have demonstrated attacks on these systems. Side-channel attacks such as the Spectre family of attacks, target access-based timing side-channels in cache subsystems [7]. Attacks focused on main memory range from cold-boot attacks [36], targeting off-chip DRAM, to timing and access pattern side-channels in both on-chip and off-chip memories.

A. *Oblivious RAM*

Oblivious RAM (ORAM) obfuscates the access pattern of a memory visible to an attacker in order to prevent information leakage. Oblivious RAM was first proposed in [37] and was built upon by [38] and [39]. The threat model for ORAMs assumes an attacker has access to the memory and bus or is otherwise able to observe the memory content, accessed addresses, and operations (read/write). However, in the threat model, attackers cannot probe the internal state of the system issuing the memory requests. Security focused SoCs have leveraged ORAM to prevent attackers with physical access to a processor and its off-chip memory from learning anything about the program being executed or the data it is executing on [9].

A variety of ORAM implementations have been proposed. One implementation, Path ORAM [39], uses a binary tree structure in the ORAM memory to split data into buckets of a small constant size. Buckets of data are encrypted to hide their contents from attackers. To obscure the bucket being accessed, whole paths of the binary tree (from root to leaf) are read and written for each operation. Path ORAM's use of the binary tree structure and the reading of whole paths result in a simple and efficient ORAM scheme suitable for hardware implementation. A memory controller on a host processor must maintain a position map to track which data is mapped to which bucket in the tree. The position map is stored in the processor's on-chip memory which is assumed to be out of reach for attackers. A relatively small amount of processor-side storage is used to hold blocks of data during read and write operations.

Path ORAM implements oblivious RAM with a required bandwidth of $O(\log N)$ for N blocks of data in the ORAM memory. The authors describe a recursive implementation of Path ORAM to reduce the required client-side storage at the cost of increased bandwidth between the client and ORAM memory. Client-side storage is reduced by storing the position map of the Path ORAM in a smaller Path ORAM memory. Position maps are recursively stored in smaller Path ORAMs until a constant sized position map can be stored on client storage.

Another implementation, described in [38], uses a randomized shell sort [40] and cuckoo hashing [41] to implement an ORAM scheme. The main advantage of the scheme presented in [38] is the $O(N)$ storage size. This storage size means that the entire ORAM memory is used to store useful data. This is not the case in Path ORAM, where some ORAM memory may be used to store the recursive position maps. Use of the entire memory comes at the expense of increased bandwidth requirements.

The complexity and overhead of current ORAM schemes often mean that ORAM cannot practically be incorporated into many systems, especially power constrained systems, such as the low-power SoCs focused on in this survey. Battery powered devices, in particular, are often susceptible to threats defended against by ORAM, i.e., an attacker with physical access to a device's memory in an uncontrolled environment. However, the inclusion of ORAM would often lead to unacceptably short intervals between battery recharges. While new ORAM implementations, such as Path ORAM [39] improve on the complexity or overhead of other implementations, more research must be completed before ORAM is suitable for most battery powered devices. Future research could focus on improving the efficiency of existing ORAM implementations or developing new techniques to obscure content and access patterns to a memory system.

B. *Memory Encryption*

Encrypting memory (either on-chip or off-chip) used by a processor offers less protection than ORAM but incurs significantly lower power, performance, and area overhead. Memory encryption is sufficient for protecting memory against an attacker capable of observing data in a memory but not its access pattern. For such attack models, either the access pattern is not considered secret or the attacker is assumed to be unable to interpret it. Attackers are usually assumed to have physical access to the to the memory.

Memory encryption is relatively simple to implement compared to existing ORAM schemes. The two techniques are compared in Figure 3. Generally, memory encryption is achieved with a hardware encryption module placed between a processing element and the vulnerable memory interface to encrypt blocks of data as they exit the security boundary and decrypt them as they enter it. AES based encryption is frequently used for main memory encryption [9] [42]. ORAM schemes require both the encryption module and an additional controller to manage the ORAM and return the appropriate read data.

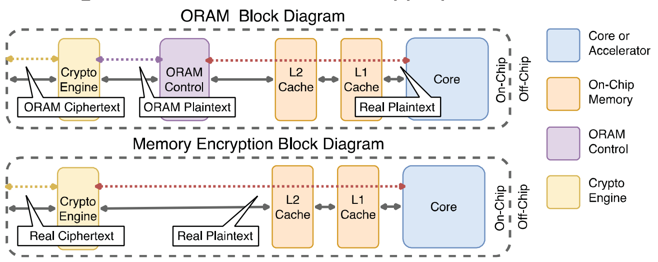

Fig. 3. ORAM scheme and memory encryption block diagrams.

The main challenge of implementing secure memory encryption is adequate protection of the encryption key. The volatile nature of most main memories means that encryption keys only need to last the duration of a system's powered on state. The ability to use a new key at each system restart removes the significant challenge of protecting encryption keys when the system is powered off. The tradeoff is that a cryptographically secure random number generator must be present in the system to generate new keys each startup. With new keys used at each startup, users are not required to know or store a password and neither is the system, removing the possibility of the password being stolen while the machine is off.

Memory encryption systems implemented by AMD utilize an isolated ARM microcontroller and hardware random number generator to generate new random keys at each system reset [42]. The generated encryption key is stored in dedicated registers. The microcontroller and encryption key are not accessible to software running on the CPU. However, safeguards must be implemented to prevent attacks on the random key generation at system startup. Any firmware used to generate or manage the keys must not be readable or writeable by untrusted parties with physical access to the device.

C. *Cache Architecture*

Memory encryption and ORAM schemes can be applied to a SoC's main memory but they do not defend against cache-based side-channels such as the Spectre family of attacks [7]. Many cache-based side-channels measure the timing of memory accesses. Information is communicated to an attacker based on the length of their memory access. One such side-channel is the Prime+Probe cache side-channel discussed in [43].

In order to execute a cache side-channel attack based on a Prime+Probe side-channel, an attacker must have the ability to execute arbitrary code on a CPU sharing a cache with the victim. No physical access is required for a successful attack. The ability to execute cache-based timing side-channel attacks without physical access makes Virtual Machines (VM) running alongside other VMs on a cloud provider's hypervisor particularly vulnerable to these attacks.

Recent research has focused on mitigating or eliminating cache-based side-channel attacks. In one such example, the authors develop a non-monopolizable (NoMo) cache architecture that partitions shared caches between threads of execution on a simultaneous multithreaded (SMT) processor [44]. This prevents a victim and attacker from evicting each other's cache line. The NoMo cache architecture reserves at least one way of the cache for each thread. However, not every way must be reserved. A 4-way cache shared between two hardware threads could reserve one way for each hardware thread and share the other two ways between them. Information cannot be leaked from reserved cache ways because an attacker cannot influence a victim's reserved cache. However, information can be leaked through the shared cache ways. Most known access-based cache timing side-channels could be completely eliminated by assigning each way to a thread, effectively creating multiple isolated, unshared caches dedicated to a hardware thread. However, in practice, this would carry a high-performance overhead, as the whole cache capacity would never be available to a single thread.

Another cache architecture to mitigate side-channel attacks is Janus [45]. Janus allows programs vulnerable to cache-based side-channel attacks to enable or disable arbitrary cache blocks to create timing and power behavior that is dynamic and difficult to extract information from. The Janus cache architecture is fully associative and uses a Least Recently Used (LRU) replacement policy. Each block in the cache is marked with an additional flag to indicate if the cache block is considered active or inactive.

When a cache block is considered inactive, it will retain its data but all memory requests to that block automatically miss. Disabling blocks in a fully associative cache gives vulnerable programs control over the effective hit rate of the cache, because all data will remain cachable while at least one cache line is enabled. The state of each cache block (active or inactive) is controlled with an activate and deactivate instruction. Programs vulnerable to side-channel attacks can execute these instructions to obscure their side-channels. The Janus cache architecture is evaluated with several benchmark programs and the variations in execution time for different activations of blocks is examined. Although it defends against different side-channels, Janus could be designed to prevent access-based cache timing side-channels, such as Prime+Probe, by requiring a high privilege level to run the custom instructions. A trusted OS could execute the activate and deactivate instructions transparently to user level programs.

Eliminating cache-based side-channel attacks is an area of active research and will undoubtedly receive continued focus in future works. While the Janus and NoMo cache architectures can significantly reduce the likelihood of a successful cache-based timing side-channel attack, stronger isolation is needed to guarantee no such side-channels exist. For cases that require such guarantees, eliminating shared caches or including a secure enclave (Section II-A) is likely necessary.

IV. Non-volatile Memory System Techniques

Embedded systems and low power SoCs are often in uncontrolled environments where an attacker could have physical access to a device's non-volatile memory (NVM). These systems store sensitive data including programs, operating systems or encryption keys in their NVM. With access to a system's NVM, an attacker could read or modify the contents, allowing them to steal keys or write malicious software to the device. To counter such attacks, researchers have developed encryption and authentication techniques for non-volatile memory. New technologies such as nanoelectro-mechanics, offer the possibility of new non-volatile memory architectures with an additional focus on security.

A. Full Disk Encryption

One of the simplest ways to protect a system's non-volatile memory is with full disk encryption. Full disk encryption prevents an attacker with physical access to a lost or stolen NVM device from reading its contents. Generally full disk encryption is handled with software, encrypting and decrypting data as it is read and written to the disk [36]. Some drives handle full disk encryption in the drive's hardware and software (firmware) freeing the host CPU from the encryption task. These drives are known as self-encrypting drives (SED). While handling encryption on a drive eliminates successful cold boot attacks on the system's main memory, these drives are often vulnerable to similar attacks on their own internal memory. The authors of [46] demonstrate that some SEDs are vulnerable to hot-plug attacks where the decryption password is given by the host PC before the SED's serial advanced technology attachment (SATA) cable is removed and attached to an attacker's system. The authors found that many SEDs do not check if the SATA cable has been removed and, therefore, remain decrypted as long as they are connected to power.

B. Limited-Use Memory

In hardware and circuit design, minimizing the effects of circuit or device wearout is often a key goal of designers. After a new technology is created, generally, researchers focus on improving the technology's endurance to create practical products. However, the authors of [47] take a different approach. They propose a design that leverages the limited endurance of nanoelectromechanics (NEMS) switches to develop devices resistant to brute force password attacks. Their threat model assumes a mobile device, such as a smart phone or other embedded system, has been fabricated and programmed by a trusted party (i.e. an attacker cannot use a backdoor to attack the device). As the device is considered mobile, an attacker is assumed to have physical access to it. The attacker's goal is to unlock the device with a brute force password attack, gaining access to its protected data.

To prevent this, the authors design NEMS switches with an endurance to match the lifespan of the device. For example, NEMS switches could be included in the read logic of the password hash memory on a smartphone. The NEMS switch circuit could be designed to support 50 reads (phone unlocks) per day for the lifetime of the device. Assuming a lifetime of five years yields 91,250 reads. Each phone unlock requires the NEMS switches to activate, increasing the chance of their failure. After the switches fail, the read logic of the memory will no longer function. Limiting the endurance of the read logic allows a user to unlock the device for the entire expected lifespan but prevents most brute force attacks on a reasonably complex password. The key challenge with designing NEMS switches for a given endurance is ensuring the minimum and maximum number of device operations is supported. Generally, the endurance of a single switch cannot be controlled with the precision needed to ensure it fails shortly after the device's end-of-life. The NEMS switches examined by the authors fail after less than 1,000 cycles for low endurance switches or after millions of cycles for high endurance switches. In order to achieve the desired minimum and maximum bounds, the authors propose the use of multiple NEMS switches. Several low endurance switches can be placed in parallel, providing functional read logic while at least one switch is still working. Alternatively, several high endurance switches can be placed in series to provide reliable read logic while all switches are functional. Figure 4 shows series and parallel implementations of NEMS switches. To further control the bounds of reliability, the authors propose using Shamir's secret-sharing scheme [48] to produce a parallel structure where *k* out of *n* switches must remain functional for the memory to be reliably read.

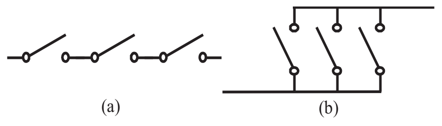

Fig. 4. Series high enduracе switches (a), parallel low endurance switches (b).

The NEMS based circuits proposed in [47] show a promising technique to root the security of non-volatile memory in the physical properties of the technology it is built with. The authors note that their research is an early work and focus on demonstrating effective control of the minimum and maximum wearout bounds for switch designs. The implementation of a memory utilizing the proposed switch designs is left for future work. More research must be completed before a practical implementation can be created.

V. Network-On-Chip Interconnect Techniques

In System-on-Chip (SoC) designs of sixteen or more processing elements, Network-on-Chip (NoC) has become the preferred communication interconnect architecture [49]. Since the interactions between processing elements in these systems happen in the NoC, it has been a fertile ground for attacks.

In general, NoC-based communication architectures are vulnerable to three types of attacks [50]. The first is On-Chip Denial of Service (OC-DoS) attacks: where a rogue PE injects a deluge of useless packets into the network, blocking another PE from accessing a needed resource or severely degrading the system performance. The second is Virtual Channel (VC) attacks: the router's VCs can be plowed, allowing malicious flows to build their packet contents out of other flows' residual data. The third is physical memory attacks: traditional security features built in the Memory Management Unit (MMU) or the Direct Memory Access (DMA) can be circumvented. In these architectures, memory modules tend to be distributed shared resources servicing both local and remote PEs. The security mechanisms implemented between the local PE and the memory module are often more defined. Remote accesses through network are more vulnerable.

Most NoC focused security features attempt to provide some level of isolation in the NoC. Given the latency sensitive nature of NoC communication, techniques such as encryption and authentication frequently carry too high an overhead for most systems, especially the low-power SoC systems focused on here.

A variety of techniques have been proposed to prevent or mitigate the NoC attacks described above. Wassel et al. implement a non-interfering scheme for secure NoC in SurfNoC [51]. Sajeesh and Kapoor have highlighted in [52] some of the advantages of implementing security policies at the network interface level in NoC based systems for secure communication among such IP cores. In [53], Porquet et al. introduce a solution for co-hosting different protection domains on the same shared memory multiprocessor SoC using a NoC architecture.

Kinsy et al. have proposed a framework for a secure many-core computing architecture where different trust level cores can be integrated onto the same chip [54] [55]. Their proposed design includes (1) a processing-element-oblivious secure network interface architecture, (2) a programmable, efficient, and distributed group key management algorithm, and (3) a hardware-supported, security-aware on-chip routing. The area overhead of the framework is 17% for the benchmarked system.

Modern SoC designs frequently leverage IP from a variety of third parties. Adding security features to the interconnect between them can prevent a malicious IP from disrupting the entire system. While the overheads of systems such as [54] and [52] can be moderate or high, the defenses they provide will become more and more important as SoCs are increasingly built with untrusted IP cores.

VI. Technique Comparison

In this section, we summarize which hardware-based security techniques can mitigate the various attacks mentioned in this survey. Secure enclaves can mitigate data theft and access-based cache side-channels with physically isolated hardware but do not prevent power-based side-channels without purpose built features such as ORAM or execution obfuscation. While execution obfuscation can mitigate on-chip side-channels, it does not directly protect off-chip memory. For that, memory encryption or ORAM are needed. Protecting non-volatile memory generally requires disk encryption, but encrypted drives can be vulnerable to brute force attacks. Memory designed for a limited number of uses could prevent such attacks if reliability challenges can be overcome.

It is clear that no single technique is adequate to defend against each of the attacks faced by low-power IoT devices. The attacks described here cover only a small portion of all possible attacks. The attacks mitigated for each technique discussed in this survey are shown in Table 1.

Modern attacks faced by computing systems have made it apparent that software based security alone is not sufficient. Systems in uncontrolled environments are vulnerable to attacks leveraging physical access, in addition to remotely exploitable side-channels and software vulnerabilities. To mitigate these threats, the security of vulnerable systems can be rooted in hardware. PUFs and limited use memory offer examples of this hardware rooted security. By combining several hardware-based security techniques in low-power SoC systems, designers can target their protection to match their threat model. Continued research focused on reducing overheads of existing techniques and the development of new ones will enable SoC designs to mitigate a larger number of attacks.

	Access-Based Cache Side-Channel	Power-Based Cache Side-Channel	Binary Reverse Engineering	In-Memory Data Theft	On-Chip Data Theft	Data at Rest Theft	Counterfeiting	On-Chip DoS
Secure Enclaves	✓			✓	✓			
Execution Obfuscation	✓	✓	✓					
PUFs							✓	
ORAM	✓	✓		✓				
Memory Encryption				✓				
Cache Arch.	✓	✓			✓			
Disk Encryption						✓		
Limited Use Memory						✓		
NoC Isolation				✓	✓			✓

TABLE I. ATTACKS MITIGATED BY EACH TECHNIQUE.

VII. Conclusion

In this survey, we have examined hardware security-based techniques suitable for low-power SoC designs. Given the uncontrolled environments that low-power SoC based embedded and mobile systems often operate in, techniques to mitigate threats based on physical access or remote side-channels were explored. Attacks faced by low-power SoCs target all aspects of the system including processing elements, volatile memory, non-volatile memory, and NoCs. As such, we have explored security techniques related to each of these major SoC subsystems. Advantages and disadvantages of each, as they relate to low-power SoCs, were discussed. Comparisons of the mitigated threats demonstrate that no single technique can defeat the numerous attacks faced by low-power SoCs deployed in uncontrolled environments. To secure their systems, designers must leverage several techniques to mitigate the most prevalent threats faced by their systems.

REFERENCES

[1] Apple, “Ios security,” apple.com/business/docs/iOS Security Guide.pdf.

[2] A. ARM, “Security technology building a secure system using trustzone technology (white paper),” ARM Limited, 2009.

[3] J. Greene, “Intel trusted execution technology,” Intel Technology White Paper, 2012.

[4] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan, “The sorcerer’s apprentice guide to fault attacks,” *Proceedings of the IEEE*, vol. 94, no. 2, pp. 370–382, Feb 2006.

[5] J. Fan and I. Verbauwhede, “An updated survey on secure ecc implementations: Attacks, countermeasures and cost,” in *Cryptography and Security: From Theory to Applications*. Springer, 2012, pp. 265–282.

[6] E. Hess, N. Janssen, B. Meyer, and T. Schu ̈tze, “Information leakage attacks against smart card implementations of cryptographic algorithms and countermeasures–a survey,” in *EUROSMART Security Conference*, vol. 130. Citeseer, 2000.

[7] P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, “Spectre attacks: Exploiting speculative execution,” *CoRR*, vol. abs/1801.01203, 2018.

[8] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, “Meltdown,” *CoRR*, vol. abs/1801.01207, 2018.

[9] C.W.Fletcher, M.v.Dijk, and S.Devadas, “A secure processor architecture for encrypted computation on untrusted programs,” in Proceedings of the seventh ACM workshop on Scalable trusted computing. ACM, 2012, pp. 3–8.

[10] C. W. Fletcher, M. Van Dijk, and S. Devadas, “Compilation techniques for efficient encrypted computation.” IACR Cryptology EPrint Archive, vol. 2012, p. 266, 2012.

[11] M. A. Kinsy, D. Kava, A. Ehret, and M. Mark, “Sphinx: A Secure Architecture Based on Binary Code Diversification and Execution Obfuscation,” Boston Area Architecture 2018 Workshop (BARC18), Feb. 2018.

[12] D. Bauder. An anti-counterfeiting concept for currency systems. Research report PTK-11990. Sandia National Labs, 1983.

[13] K. Lofstrom, W. R. Daasch, and D. Taylor. Ic identification circuit using device mismatch. In 2000 IEEE International Solid-State Circuits Conference. Digest of Technical Papers (Cat. No.00CH37056), pages 372–373, Feb 2000

[14] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas. Silicon physical random functions. In Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS ’02, pages 148–160, New York, NY, USA, 2002.

[15] NXP Strengthens SmartMX2 Security Chips with PUF Anti-Cloning Technology.
http://www.nxp.com/news/press-releases/2013/02/nxp-strengthens-smartmx2-security-chips-withpuf-anti-cloning-technology.html.

[16] E. Times. NXP Strengthens SmartMX2 Security Chips with PUF Anti-Cloning Technology. UBM Tech Electronics, 2010. (Referenced on page D-6.)

[17] J. Delvaux, R. Peeters, D. Gu, and I. Verbauwhede. A survey on lightweight entity authentication with strong pufs. ACM Comput. Surv., 48(2):26:1–26:42, Oct. 2015.

[18] R. Pappu, B. Recht, J. Taylor, and N. Gershenfeld. Physical one-way functions. Science, 297(5589):2026–2030, 2002.

[19] B. Gassend, M. V. Dijk, D. Clarke, E. Torlak, S. Devadas, and P. Tuyls. Controlled physical random functions and applications. ACM Trans. Inf. Syst. Secur., 10(4):3:1–3:22, Jan. 2008.

[20] M. Tehranipoor and C. Wang. Introduction to Hardware Security and Trust. Springer Publishing
Company, Incorporated, 2011.

[21] J. Guajardo, S. Kumar, G. Schrijen, and P. Tuyls. FPGA intrinsic PUFs and their use for IP protection. In Cryptographic Hardware and Embedded Systems (CHES), pages 63–80, 2007.

[22] D. Holcomb, W. Burleson, and K. Fu. Initial SRAM state as a fingerprint and source of true random numbers for RFID tags. In Proceedings of the Conference on RFID Security, 2007.

[23] U. Ruhrmair, S. Devadas, and F. Koushanfar. Security based on physical unclonability and disorder. In M. Tehranipoor and C. Wang, editors, Introduction to Hardware Security and Trust, chapter 4, pages 65–102. Springer, 2012.

[24] B. Skoric, S. Maubach, T. Kevenaar, and P. Tuyls. Information-theoretic analysis of capacitive physical unclonable functions. Journal of Applied Physics, 100(2):024902, 2006.

[25] P. Tuyls, G.-J. Schrijen, B. ˇSkori´c, J. van Geloven, N. Verhaegh, and R. Wolters. Read-proof hardware from protective coatings. In Proceedings of the 8th International Conference on Cryptographic Hardware and Embedded Systems, CHES’06, pages 369–383, Berlin, Heidelberg, 2006. Springer-Verlag.

[26] D. Holcomb, W. 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.

[27] D. Karakoyunlu and B. Sunar. Differential template attacks on puf enabled cryptographic devices. In 2010 IEEE International Workshop on Information Forensics and Security, pages 1–6, Dec 2010.

[28] A. Mahmoud, U. R¨uhrmair, M. Majzoobi, and F. Koushanfar. Combined modeling and side channel attacks on strong PUFs. Cryptology ePrint Archive, Report 2013/632, 2013.

[29] U. R¨uhrmair, F. Sehnke, J. S¨olter, G. Dror, S. Devadas, and J. Schmidhuber. Modeling attacks on physical unclonable functions. In Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS ’10, pages 237–249, New York, NY, USA, 2010. ACM.

[30] C. Helfmeier, C. Boit, D. Nedospasov, and J.-P. Seifert. Cloning physically unclonable functions. In IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), pages 1–6, 2013.

[31] D. Nedospasov, J.-P. Seifert, C. Helfmeier, and C. Boit. Invasive puf analysis. In Proceedings of the 2013 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC ’13, pages 30–38, Washington, DC, USA, 2013. IEEE Computer Society.

[32] G. Hospodar, R. Maes, and I. Verbauwhede. Machine learning attacks on 65nm arbiter pufs: Accurate modeling poses strict bounds on usability. In 2012 IEEE International Workshop on Information Forensics and Security (WIFS), pages 37–42, Dec 2012. (Referenced on page D-7.)

[33] U. Ruhrmair, J. Solter, F. Sehnke, X. Xu, A. Mahmoud, V. Stoyanova, G. Dror, J. Schmidhuber, W. Burleson, and S. Devadas. Puf modeling attacks on simulated and silicon data. Trans. Info. For. Sec., 8(11):1876–1891, Nov. 2013.

[34] G. T. Becker. Robust fuzzy extractors and helper data manipulation attacks revisited: Theory vs practice. Cryptology ePrint Archive, Report 2017/493, 2017. http://eprint.iacr.org/2017/493.

[35] J. Delvaux and I. Verbauwhede. Side channel modeling attacks on 65nm arbiter pufs exploiting cmos device noise. In 2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), pages 137–142, June 2013.

[36] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten, “Lest we remember: Cold-boot attacks on encryption keys,” Commun. ACM, vol. 52, no. 5, pp. 91–98, May 2009.

[37] O. Goldreich and R. Ostrovsky, “Software protection and simulation on oblivious rams,” Journal of the ACM (JACM), vol. 43, no. 3, pp. 431–473, 1996.

[38] B. Pinkas and T. Reinman, “Oblivious ram revisited,” in Annual Cryptology Conference. Springer, 2010, pp. 502–519.

[39] E. Stefanov, M. Van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas, “Path oram: an extremely simple oblivious ram protocol,” in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 299–310.

[40] M. T. Goodrich, “Randomized shellsort: A simple oblivious sorting algorithm,” in *Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA ’10. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2010, pp. 1262–1277.

[41] R. Pagh and F. F. Rodler, “Cuckoo hashing,” *Journal of Algorithms*, vol. 51, no. 2, pp. 122 – 144, 2004.

[42] D. Kaplan, J. Powell, and T. Woller, “Amd memory encryption,” *White paper*, 2016.

[43] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, “Last-level cache side-channel attacks are practical,” in *2015 IEEE Symposium on Security and Privacy*, May 2015, pp. 605–622.

[44] L. Domnitser, A. Jaleel, J. Loew, N. Abu-Ghazaleh, and D. Ponomarev, “Non-monopolizable caches: Low-complexity mitigation of cache side channel attacks,” *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 8, no. 4, p. 35, 2012.

[45] H. Hosseinzadeh, M. Isakov, M. Darabi, A. Patooghy, and M. A. Kinsy, “Janus: An uncertain cache architecture to cope with side channel attacks,” in *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, Aug 2017, pp. 827–830.

[46] T. Muller, T. Latzo, and F. C. Freiling, “Self-encrypting disks pose self-decrypting risks,” in *the 29th Chaos Communinication Congress*, 2012, pp. 1–10.

[47] Z. Deng, A. Feldman, S. A. Kurtz, and F. T. Chong, “Lemonade from lemons: Harnessing device wearout to create limited-use security architectures,” *SIGARCH Comput. Archit. News*, vol. 45, no. 2, pp. 361–374, June 2017.

[48] A. Shamir, “How to share a secret,” *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[49] N. E. Jerger and L.-S. Peh, “On-chip networks,” Synthesis Lectures on Computer Architecture, vol. 4, no. 1, pp. 1–141, 2009.

[50]] L. Fiorin, C. Silvano, and M. Sami, “Security aspects in networks-on-chips: Overview and proposals for secure implementations,” in Digital System Design Architectures, Methods and Tools, 2007. DSD 2007. 10th Euromicro Conference on, Aug 2007, pp. 539–542.

[51] H. M. G. Wassel, Y. Gao, J. K. Oberg, T. Huffmire, R. Kastner, F. T. Chong, and T. Sherwood, “Surfnoc: A low latency and provably non-interfering approach to secure networks-on-chip,” in Proceedings of the 40th Annual International Symposium on Computer Architecture, ser. ISCA ’13. New York, NY, USA: ACM, 2013, pp. 583–594.

[52] Sajeesh, K.; Kapoor, H.K. An Authenticated Encryption Based Security Framework for NoC Architectures. In Proceedings of the 2011 International Symposium on Electronic System Design, Kochi, India, 19–21 December 2011; pp. 134–139.

[53] Porquet, J.; Greiner, A.; Schwarz, C. NoC-MPU: A secure architecture for flexible co-hosting on shared memory MPSoCs. In Proceedings of the 2011 Design, Automation & Test in Europe, Grenoble, France, 14–18 March 2011; pp. 1–4.

[54] M. A. Kinsy, S. Khadka, M. Isakov and A. Farrukh, "Hermes: Secure heterogeneous multicore architecture design," 2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), McLean, VA, 2017, pp. 14-20.

[55] M. A. Kinsy, L. Bu, M. Isakov and M. Mark: "Designing Secure Heterogeneous Multicore Systems from Untrusted Components". Cryptography, vol. 2, iss. 3, no. 12, 2018.