A Short Survey at the Intersection of Reliability and Security in Processor Architecture Designs

Lake Bu, Miguel Mark, and Michel A. Kinsy
Adaptive and Secure Computing Systems (ASCS) Laboratory
Department of Electrical and Computer Engineering, Boston University

***Abstract*—Over the next decade, processor design will encounter a number of challenges. The ongoing miniaturization of semiconductor manufacturing technologies that has enabled the integration of hundreds to thousands of processing cores on a single chip is pushing the limits of physical laws. The fabrication process has also grown more complex and globalized with widespread use of third-party IPs (intellectual properties). This development ecosystem has complicated the security and trust view of processors. Some of the pressing processor architecture design questions are: (1) how to use reconfiguration and redundancy to improve reliability without introducing additional and potentially insecure system states, (2) what analytical models lend themselves best to the joint implementation of reliability and security in these systems, and (3) how to optimally and securely share resources and data among processing elements with high degree of reliability. In this work, we present and discuss (1) principal reliability approaches - error correction code, modular redundancy, (2) processor architecture specific reliability, (3) major secure processor architectures. We also highlight key features of a small representative class of the secure and reliable architectures.**

I. Introduction

The intersection of reliability and security in the design of processor architectures is now a critical concern in a wide range of embedded computing, communications systems, and connected devices. On one hand, as feature size shrinks, transistors become less reliable and component failures increase. Transistor scaling and integration result in reliability challenges, including interference from electric fields, shrinking of the maximum-minimum voltage window, thermo-mechanical limitations, and soft, transient and intermittent errors. On the other hand, the emergence of general-purpose system-on-chip (SoC) architectures has given rise to a number of significant security challenges. The current trend in SoC design is system-level integration of heterogeneous technologies consisting of a large number of processing elements such as programmable RISC cores, memories, DSPs, and accelerator function units/ASIC. These processing elements may come from different providers, and application executable code may have varying levels of trust.

In this short survey, we attempt to highlight some of the pressing processor architecture design questions:

1) **Reliability Issues:** how reconfiguration and redundancy are used to improve reliability without introducing additional and potentially insecure system states;
2) **Security Issues:** how to optimally and securely share resources and data among processing elements which have different levels of trust;
3) **Security and Reliability in Architecture:** what analytical models lend themselves best to the joint implementation of reliability and security in these systems.

Over the years, there have been many attempts to address the aforementioned processor architecture design issues. Some commonly accepted approaches and methodologies have even emerged. In this work, we define "reliability" as the property of keeping the system in a pre-defined/desired/accepted functional condition. "Security" is characterized by the capability to protect the system from malicious attempts which either drive the system away from the accepted functional conditions, or exploit the limitations and restrictions of the system. These attempts can be either invasive or non-invasive.

On the topic of reliability, we first emphasize general approaches: error control codes and their application in processor architecture designs, modular redundancy for dependable functionalities in architectures, and processor architecture specific reliability methods such as ACE (architecturally correct execution), and AVFs (architectural vulnerability factors). We then discuss the existing and potential vulnerabilities of these approaches.

As for the security aspect, we start with major commercially available and academic secure processor architectures, such as Intel's Software Guard Extensions (SGX) and Trusted Execution Technology (TXT), ARM TrustZone Technology and derived processor architectures, MIT Aegis Secure Processor, IBM 4765 Secure Coprocessor, and Apple Secure Enclave Processor (SE). We then examine work on privacy and permission management targeting heterogeneous multicore systems. Finally, we stress the potential vulnerabilities and attacks on some of these security-aware architectures.

Our focus on the security issues in heterogeneous computing environments primarily centers around multicores where the cores may have different levels of trustworthiness. The problem on such compute systems is how to optimally and securely share resources and data among those processing units, while maintaining individual tenant security and preventing data leakage among the units.

The rest of the paper is organized as follows: section II is on the reliability-oriented architecture designs, followed by their vulnerabilities. Section III is on the security-oriented architecture designs, and the existing and potential vulnerabilities. Section IV discusses the joint design and implementation of reliability and security. Section V concludes the paper.

2159-3477/18/$31.00 ©2018 IEEE
DOI 10.1109/ISVLSI.2018.00031

II. Reliability-Aware Processor Architectures and Designs

In this section we will first introduce the error control codes (ECCs) as a universal technique to provide reliability in architecture designs. Besides being used as a tool to preserve the data integrity, the mathematical principle of ECCs can also be leveraged in the design of reliable systems or processors. Besides introducing the redundant modules using ECC, the processor architecture specific reliability such as ACE (architecturally correct execution), and AVFs (architectural vulnerability factors) are also presented. The vulnerabilities of current reliability techniques will be discussed in the final subsection.

A. ECC-based Reliable Processor Design

The error control codes (ECCs) are usually used to prevent data from being distorted by random errors. The most straightforward applications are the reliable buses, memories, and caches [1, 2]. We define the following notations:

- x: the original source data;
- y: the redundant portion computed based on x;
- v the encoded codewords of x;
- e: the random errors on v;
- $G()$ the generating function of v;
- $H()$: the checking function of v;
- $f()$: the functional module's function;
- $P()$: the predictor function for $f()$;
- $\sim$: the distortion symbol.

We introduce three of the most common ways that ECCs can be used to assist the reliability of a system.

1) Random Error Correction: the procedure that an ECC module uses to protects data from random errors is as follows:

i. Before a piece of data x is transmitted or stored in a system, it is first encoded by a generating function that $G(x) = v$, where x should be able to be retrieved from v;
ii. During the transmission or storage of v, it might be corrupted by random errors e, that $v + e = \tilde{v}$;
iii. When the piece of data is to be extracted, a decoding function is used to retrieve the correct piece: $H(\tilde{v}) = v$. And so x can be derived from v.

The work flow of such procedure is shown in Fig. 1.

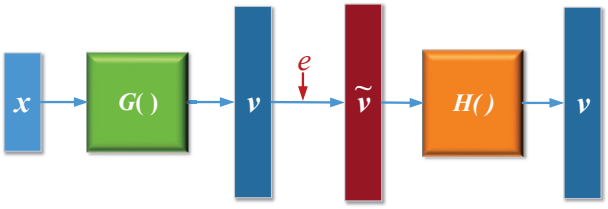

Fig. 1. In systematic encoding, $v = x||y$ and y is the redundant portion computed by $G(x)$ based on x, $||$ the concatenation operator. In this case the value of x is obvious once the correct v is retrieved. In non-systematic encoding, x has to be computed through certain algorithms from v.

2) Data Regeneration: This use of ECC modules is similar to Fig. 1, except that instead of v being distorted by a random error e, now part of it is missing. Thus the $H()$ is used to regenerate v rather than removing e. Due to the property of ECC, when used for data re-generation, it usually has a stronger capability in fault tolerance. This approach is now popular in machine learning acceleration [3] and heterogeneous clusters' straggler tolerance [4].

3) Self-checking Checkers: There have been many research efforts on the application of ECCs to the circuits or functional modules as the self-checking checkers (SCC) to verify the correctness of their functionality. The common thread in these efforts is the addition of a parallel module named the "predictor" to the original function module, which generates the corresponding check bits at the same time of the functional module's output. The predictor's check bits and the functional module's output are verified by the decoder for error detection, or correction. Together, the predictor and decoder form an SCC system. The procedure of a SCC's self-correction is as follows:

i. When an input x comes into a functional module $f()$, it is also fed into a predictor module $P()$, where $P()$ is a combination of $f()$ and $G()$;
ii. During the computation of $f(x)$, which is the system's original functionality, $P(x)$ is also computed. Either module can be malfunctional;
iii. The decoder verifies $f(\tilde{x})$ and $P(\tilde{x})$ and outputs the correct $f(x)$ to maintain reliability.

Figure 2 illustrates the workflow of a SCC.

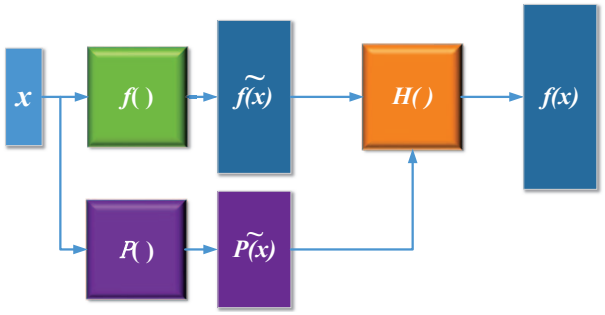

Fig. 2. Instead of random error correction on the source data, the goal of a SCC is to protect the functionality $f()$ of a given system or computation node. With proper optimization, $P()$ does not necessarily have higher complexity than $f()$;

Various codes have been devised as ECCs such as the repetition codes, cyclic codes, Hamming codes, and Reed-Solomon (RS) codes [5, 6, 7]. They are characterized by different levels of error tolerance capability and decoding complexity. In recent years, low density codes have become more popular due to their low complexity in decoding [8, 2].

It should be noted that duplication or triplication systems in processor designs have originated from ECCs. In a duplication system [9], two systems with identical functionality will perform the same operations, and the results of which will be compared. A triplication system [10], involves three identity function whose results will participate in a majority voting

to tolerate the malfunction of a single system. These two techniques leverage the concept of repetition codes in ECCs.

B. Architecturally Correct Execution (ACE), and Architectural Vulnerability Factors (AVFs)

Researchers from Intel [11] introduced the concept of architecturally correct execution (ACE). In their definition, a bit in a system is related to architecturally correct execution (ACE) if it affects the output of the program. Other bits which do not have such influence are called un-ACE bits. A structure's architectural vulnerability factor (AVF) is defined as the probability that a fault in the structure will result in an erroneous output. One of the fundamental differences between AVF estimation and ECC is, the former is more of a methodology to evaluate an architecture's reliability, and the latter is a practical technique to ensure the dependability of an architecture. Also, the former tracks the bits with an impact to the final outcome only (particularly from the user's perspective), while the latter tries to treat all the bits equally.

A program running on a faulty architecture has multiple possible outcomes. There can be faults resulting no error, silent data corruption (SDC), and detected but unrecoverable errors (DUE). The correlation among them and an architecture's error tolerance capability are given by [12] and depicted in Figure 3.

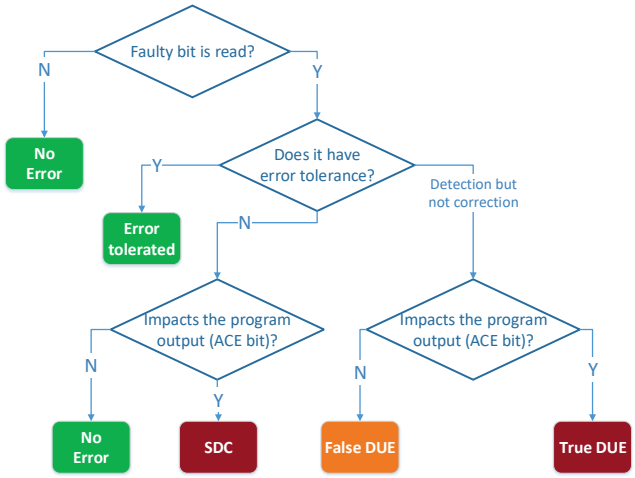

Fig. 3. In this diagram, there can be undetected and uncorrected faults in an architecture, but they do not necessarily affect the final outcome. The "detection but not correction" branch can be the scenarios that the architecture is only equipped with parity check codes but not any ECC with Hamming distance larger than 3. Or it can be that the architecture only has an error detection subsystem but not an error correction subsystem, such as the duplication subsystems.

In [11], authors proposed an efficient approach for estimating AVFs that uses only a subset of the processor state bits. The used bits in a processor state storage cell/structure are the ones related to ACE. They will show a distortion in the output when an error occurs. Other bits in storage cells are the un-ACE bits, which can be flipped without causing a functional error. The authors provide the equations and different approaches for computing the AVFs.

1) Statistical Fault Injection: In this test scheme, random errors are injected in both randomized space and time domains. The results will then be compared with a pre-generated reference result set, or an error-free model. The AVF is computed by the fraction of mismatches divided by the total number of injected errors.

If there is no mismatch observed, it can be because either the error is tolerated, or is masked (silent errors). The latter is a more complicated situation and has to be studied by a complete comparison in system states [13].

2) Little's Law: This method is suitable for the early stage of a design before the RTL is generated. Denote N as the average number of bits in the architecture, B the average bandwidth per cycle into the architecture, and L the average latency a bit through the architecture, the subscript ACE for the ACE bits. Then the AVF can be estimated by:

$$\frac{B_{ACE} \times L_{ACE}}{N}$$

3) ACE Analysis in Performance Models: In this method, a performance model is used to determine which bits are ACE and which are un-ACE. A conservative assumption is made that, a bit is ACE unless it is proved as an un-ACE. This methodology can be more time efficient than others.

C. Vulnerabilities of Reliability-Aware Architecture Designs

For most reliability-oriented designs, there can be a large number of "invisible" errors never detected by the system. The invisibility is not due to the lack of error detection or correction subsystems, but because of their linearity. We will firstly introduce the concept of the "kernel" as a measurement of the number of invisible errors in an architecture.

Definition 1. Suppose C is the set of N-bit ECC codewords and $H()$ is the decoding/error detection/error correction function. C is defined by $C = \{v|H(v) = 0\}$. Set K_d is called the Kernel of C if:

$$K_d = \{e|e + v \in C, \ \forall v \in C\}.$$

Under this definition, if $H()$ is a linear function (which is the case for most architectures), and there exist an error e that $H(e) = 0$, then we have:

$$H(\tilde{v}) = H(e + v) = H(e) + H(v) = 0 + 0 = 0. \quad (1)$$

Then this error is invisible for $\forall v \in C$. If C is linear, then the set of e which is the kernel $K_d = C$. This result shows that for any architecture with a linear error control function, there exists a large number of invisible errors. The good news is that for most systems, those invisible errors are more than one bit, which can be very rare. Thus most single-bit errors can still be taken care by the SEC-DED subsystems. However this potential vulnerability can still be leveraged by attackers to inject forever-masked errors.

III. Security-Aware Processor Architectures and Designs

Security, unlike reliability, is a much larger and more complicated topic for all architecture designers. Different architectures targeting different security demands will end up with very distinct structures. Therefore in this section, instead of giving a universal design methodology, we will present a number of commercialized and representative security-aware architecture designs, as well as their advantages and vulnerabilities. The subsections will include the introduction of the MIT Aegis Secure Processor, the Apple Secure enclave processor (SEP), the ARM TrustZone technology, and the IBM 4765 Secure Processor.

A. MIT Aegis Secure Processor

Aegis [14] is a secure processor which aims to provide conventional software-based authentication and addresses a critical assumption made by other secure processor implementations: physical attacks are infeasible or meaningless. Its architectural design philosophy is based on the premise that only the Aegis processor can be authenticated and trusted. External components such as non-volatile memory and other processors are treated as non-trustworthy by default. The core of Aegis's protection is centered on Silicon Physical Random Functions (SPUFs) which leverage unique timing delays in integrated circuits created by the semiconductor manufacturing process [15]. Aegis uses this unique characteristic in the form of a PUF delay circuit which is used for secret key generation and authentication. Furthermore, restricting protection to one chip prevents the leakage of secrets through unsecured communication between multiple processing units.

To protect software, Aegis first introduces four additional processing modes: Standard (STD), Suspended Secure Processing (SSP), Tamper-Evident (TE) and Private Tamper-Resistant (PTR). STD and SSP are the lowest privilege mode which has no access to private memory and can only enter the more secure TE and PTR mode. TE has read/write access to verified memory and a subset of security functions. PTR mode is the most privileged due to its access to PUF instructions. Second, software can be authenticated using an authentication scheme with SPUFs such as a public/private key protocol. Lastly, off-chip memory protection in the form of Integrity verification (IV) and Memory Encryption (ME) can be enabled when the supervisor switches the processor into TE or PTR mode after boot. IV and ME aim to provide defense against both software and hardware attacks. To accomplish this, the processor partitions the available memory into IV and ME regions which can overlap. IV protects regions through detecting and preventing any unintended modifications and ME utilizes encryption to hide sensitive contents. A trusted supervisor, such as a kernel manages the sharing of these protected regions. Later on they also proposed a version of Aegis which is resistant to malicious operating systems [16].

With these features combined, the Aegis secure one chip processor can defend against a wide range of attacks. Brute force based attacks are not feasible due to the sheer number of challenge-response pairs that can be generated. Attackers may then attempt to create a timing model of the PUF delay circuit but this is not possible since no information is leaked from the circuit. Likewise, an attacker cannot duplicate the PUF circuit due to nature of the manufacturing process. Even if the attacker gains physical access to the processor and tries to probe timing information, the data collected will be useless due to the interference caused by the probe.

Although the authors noted the omission of side-channel attacks and learning attacks to the PUF which is the fundamental source of security, overall Aegis can provide a strong defense with negligible overhead in gate size and performance.

B. Apple Secure Enclave Processor (SEP)

Apple's Secure Enclave Processor (SEP) [17] is a flashable coprocessor which utilizes memory encryption and hardware number generation to carry out cryptographic functions for the main processor. In a sense, SEP creates a logical wall between software and sensitive security functions so that untrusted software cannot gain access to sensitive data such as fingerprints and keys. To achieve most of its functionality, a trusted micro-kernel runs on top of the processor, sporting its own drivers and services. Given the nature of this technology, Apple has prevented the dissemination of the technical details of the processor. Therefore, technical details are only available through efforts of reverse engineering.

The basic architectural design of SEP is the separation of computation into two processors: Application processor (AP) and SEP. SEP contains completely separated hardware such as a hardware number generator, boot ROM, and crypto engine. Despite this aggressive separation, SEP is still a 32-bit processor which 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. With this mailbox, the architecture acts as a walled garden which is called the KF filter. The KF filter encapsulates and guards many of the SEP's unique hardware components. Therefore all data originating from the SoC passes through the filter and must go through the secure mailbox. Once SEP has initialized secure memory regions, it is protected from software-based attacks. To protect against physical-based attacks such as memory probing, SEP utilizes memory encryption in the form of AES-ECB, AES-CBC and AES-XEX. Furthermore, after initialization, applications which wish to interact with the encrypted data guarded by SEP must use a Bootstrap server which can enforce access and privilege rules for different functionalities such as a secure key generation service.

Overall the nature of the SEP defends against an attack model in which an attacker can compromise system software such as the kernel. However, there have been reports in 2017 that hackers have decrypted the SEP's firmware and published its secret key [18]. Although this breach does not leak any user's information or data, it makes a way for researchers and hackers to explore the vulnerabilities of SEP.

C. ARM TrustZone technology

The ARM TrustZone technology [19] is a single core secure processor technology that uses a security approach similar to that of Apples Secure Enclave processor. Its design philosophy is based on levels of trust which aims to minimize the attack surface at lower levels. In a sense, ARM TrustZone uses separation based on the concept of least privilege; software or hardware should only have access 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 normal world contains everything else. This allows an establishment of a chain of trust. Separation of zones starts with the partitioning of memory into secure and non-secure memory regions.

Naturally, through separation, non-secure world processes cannot access secure content but secure-world processes can access both secure and non-secure content. Modules called the Security Attribution Unit (SAU) and Implementation Defined Attribution Unit (IDAU) work together to determine if a memory region is secure. TrustZone also provides a subtype of secure memory, non-secure callable memory, which is an executable region which allows non-secure instructions to branch into a secure memory using Secure Gateway (SG) instructions. Despite this aggressive separation model, communication between non-secure world and secure world processes is possible via a Secure Monitor Call (SMC). Through providing these primitives for processes, TrustZone removes the need for a separate security processor that would inevitably increase the attack surface. ARM designed TrustZone as a configurable platform that can better adapt to different attack models. Specifically, TrustZone provides SoC designers with various TrustZone enabled IP modules that allow an embedded device to be tailored to a particular attack model. One particular weakness of Trustzone is that assumes that secure mode processes can always be trusted.

D. IBM 4765 Secure Processor

The IBM 4765 [20] is a secure co-processor which is placed on a PCIe card. Equipped with a hardware number generator it provides tamper-proof storage of sensitive data and cryptographic operations for activities such as SSL private key transactions. Like most secure processors, the 4765 supports several cryptographic algorithms: SHA-256, HMAC, and RSA. Due to the nature of PCIe, the 4765 is, unfortunately, an easy target for both theft and physical manipulation. Fortunately, a hardware-based tamper-proof module is included which is certified for meeting the Federal Information Processing Standard (FIPS) 1402-2 level 4 security requirements.

The tamper-proof module can detect physical abnormalities such as voltage spikes and temperatures variances and mark them as physical attacks. As a response, the tamper circuit will automatically zero out secrets stored in onboard memory devices such as Battery-Backed Static Random Access Memory (BBRAM). The tamper-proof protocol can also be made to zero out memory in the case of ejection from the PCIe slot.

On the software side, applications can interface with the 4765 through Miniboot: software internal to the module that exposes functionality. The commands available for applications depends on Miniboot's current boot level. Because Miniboot runs at boot time, its progress through different levels depends on the success of the host's power-on-self-test (POST). For example, if POST1 fails, an application would have had only access to the set of Miniboot0 commands and queries. Communication between the two available Miniboot levels, Minitboot0 and Miniboot1, and applications involve authentication using a public/private key protocol. Authentication of each command is based on the concept of roles which is essentially an Access Control Layer for various functions and services provided by Miniboot.

E. Intel Trusted Execution Technology (TXT)

Intel's Trusted Execution Technology (TXT) [21] is a hardware-based technology to examine the authenticity of the operating system and its running environment. It also relies on the Trusted Platform Module (TPM) to provide functionalities such as secure storage. The purpose of the TXT is to provide a trusted way for loading and executing system software, e.g. Operating System kernel or Virtualization Machine Monitor (VMM), even on machines with malicious software and malware. The technology supports both a static chain of trust and a dynamic chain of trust. The static chain of trust starts when the platform powers on (or the platform is reset), which resets all PCRs to their default value. For server platforms, the first measurement is made by hardware (i.e., the processor) to measure a digitally signed module (called an Authenticated Code Module or ACM) provided by the chipset manufacturer. The processor validates the signature and integrity of the signed module before executing it. The ACM then measures the first BIOS code module, which can make additional measurements. However, there have been works showing that the TXT only provides launch-time protection, but not runtime [22], against attacks such as buffer overflow etc. More importantly, researchers have been able to infect the system management mode (SMM), which is one of the most privileged software loaded on a computer, to bypass the TXT's launch examination and conduct attacks. Another research [23] shows that attackers can infect the boot loader and execute their own code before the TXT's SENTER instructions are executed.

IV. Secure Heterogeneous Multicore Architecture

It is worth stressing that the technique and technology challenges encountered in the design of single-core or homogeneous multicore secure processors are further amplified in the security-aware design of heterogeneous multicore architectures. On those systems, different cores or processing units may have different levels of trust or privacy. Thus the secure data and permission management has to be taken into consideration. Researchers in [24] and [25] have proposed

different approaches to address this issue. In [25], the authors introduced the "Hermes" architecture, which embeds its security features in the on-chip interconnect fabric. Hermes is a secure multicore computing architecture framework. It reduces the system attack surface by creating a virtualization layer that isolates compute threads based on system and user defined trust levels and security policies. Figure 4 shows a set of applications with mixed security being mapped onto a mixed security hardware. It achieves both hardware and software views of secure processing by grouping processors into physical zones called *wards* and virtual logical zones called *islands*. So that although different cores are located and operated in different manners, they are categorized to certain standardized security levels to be given corresponding permissions and privileges.

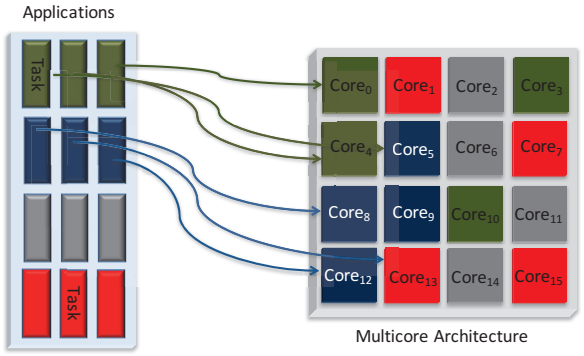

Fig. 4. Trusted/untrusted applications running on trusted/untrusted cores. Different trust levels are illustrated by different colors (e.g., red represents the least trusted program or core).

V. Conclusion

In this paper we explore and discuss the key details of reliability and security-aware processor architectures and designs. There are some established approaches to designing and evaluating reliable architectures such as ECC, ACE, and AVF. The application of error correction protection can guard processors against random and limited errors. However, the problem is more complicated when it comes to security-aware architectures. There are different security demands and attack models for each design on the market. Additionally, we present a small representative set of secure processor architectures - commercially available and academic and discuss their vulnerabilities. Finally, we briefly touched on a secure heterogeneous multicore architecture, which aims to provide a trustworthy data and permission management among heterogeneous cores with different levels of trust.

VI. Acknowledgments

This research is partially supported by the NSF grant (No. CNS- 1745808).

References

[1] C. Wilkerson, A. Alameldeen, and Z. Chishti, "Scaling the memory reliability wall." *Intel Technology Journal*, vol. 17, no. 1, 2013.

[2] P. Reviriego, S. Pontarelli, A. Evans, and J. A. Maestro, "A class of sec-ded-daec codes derived from orthogonal latin square codes," *IEEE transactions on very large scale integration (vlsi) systems*, vol. 23, no. 5, pp. 968–972, 2015.

[3] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Transactions on Information Theory*, vol. 64, no. 3, pp. 1514–1529, 2018.

[4] A. Reisizadeh, S. Prakash, R. Pedarsani, and S. Avestimehr, "Coded computation over heterogeneous clusters," in *Information Theory (ISIT), 2017 IEEE International Symposium on*. IEEE, 2017, pp. 2408–2412.

[5] L. Breveglieri, I. Koren, and P. Maistri., "Incorporating error detection and online reconfiguration into a regular architecture for the advanced encryption standard," *Defect and Fault Tolerance in VLSI Systems, 2005. DFT 2005. 20th IEEE International Symposium on. IEEE*, 2005.

[6] C.-H. Yen and B.-F. Wu, "Simple error detection methods for hardware implementation of advanced encryption standard," *IEEE transactions on computers*, 2006.

[7] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri, and V. Piuri, "A parity code based fault detection for an implementation of the advanced encryption standard," *Defect and Fault Tolerance in VLSI Systems*, 2002.

[8] L. Dai, B. Wang, Y. Yuan, S. Han, I. Chih-Lin, and Z. Wang, "Non-orthogonal multiple access for 5g: solutions, challenges, opportunities, and future research trends," *IEEE Communications Magazine*, vol. 53, no. 9, pp. 74–81, 2015.

[9] J. Johnson, W. Howes, M. Wirthlin, D. L. McMurtrey, M. Caffrey, P. Graham, and K. Morgan, "Using duplication with compare for on-line error detection in fpga-based designs," *Aerospace Conference*, 2008.

[10] P.-T. Huang, W.-L. Fang, Y.-L. Wang, and W. Hwang., "Low power and reliable interconnection with self-corrected green coding scheme for network-on-chip," *Second ACM/IEEE International Symposium on Networks-on-Chip*, 2008.

[11] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin, "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor," in *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*. IEEE, 2003, pp. 29–40.

[12] S. S. Mukherjee, J. Emer, and S. K. Reinhardt, "The soft error problem: An architectural perspective," in *High-Performance Computer Architecture, 2005. HPCA-11. 11th International Symposium on*. IEEE, 2005, pp. 243–247.

[13] N. J. Wang, J. Quek, T. M. Rafacz, and S. J. Patel, "Characterizing the effects of transient faults on a high-performance processor pipeline," in *Dependable Systems and Networks, 2004 International Conference on*. IEEE, 2004, pp. 61–70.

[14] G. E. Suh, C. W. O'Donnell, and S. Devadas, "Aegis: A single-chip secure processor," *Information Security Technical Report*, vol. 10, no. 2, pp. 63–73, 2005.

[15] G. E. Suh, C. W. O'Donnell, I. Sachdev, and S. Devadas, "Design and implementation of the aegis single-chip secure processor using physical random functions," in *ACM SIGARCH Computer Architecture News*, vol. 33, no. 2. IEEE Computer Society, 2005, pp. 25–36.

[16] G. E. Suh, D. Clarke, B. Gassend, M. Van Dijk, and S. Devadas, "Aegis: architecture for tamper-evident and tamper-resistant processing," in *ACM International Conference on Supercomputing 25th Anniversary Volume*. ACM, 2014, pp. 357–368.

[17] Apple, "Ios security," apple.com/business/docs/iOS_Security_Guide.pdf.

[18] M. Mimoso, "Hacker publishes ios secure enclave firmware decryption key," in *Threatpost*, 2017.

[19] A. ARM, "Security technology building a secure system using trustzone technology (white paper)," *ARM Limited*, 2009.

[20] T. W. Arnold, C. Buscaglia, F. Chan, V. Condorelli, J. Dayka, W. Santiago-Fernandez, N. Hadzic, M. D. Hocker, M. Jordan, T. Morris *et al.*, "Ibm 4765 cryptographic coprocessor," *IBM Journal of Research and Development*, vol. 56, no. 1.2, pp. 10–1, 2012.

[21] J. Greene, "Intel trusted execution technology," *Intel Technology White Paper*, 2012.

[22] R. Wojtczuk and J. Rutkowska, "Attacking intel trusted execution technology," *Black Hat DC*, 2009.

[23] R. Wojtczuk, J. Rutkowska, and A. Tereshkin, "Another way to circumvent intel trusted execution technology," *Invisible Things Lab*, 2009.

[24] H. Kondo, S. Otani, M. Nakajima, O. Yamamoto, N. Masui, N. Okumura, M. Sakugawa, M. Kitao, K. Ishimi, M. Sato *et al.*, "Heterogeneous multicore soc with sip for secure multimedia applications," *IEEE Journal of solid-state circuits*, vol. 44, no. 8, pp. 2251–2259, 2009.

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