Survey of Secure Processors

Suman Sau, Jawad Haj-Yahya, Ming Ming Wong, Kwok Yan Lam, Anupam Chattopadhyay
NTU, Singapore

***Abstract*—The convergence of information technology systems, data networks, embedded systems and Internet-of-Things (IoT) within the cyber-physical system (CPS) paradigm has led to the emergence of new security threats associated with the system hardware. To control risks of potential hardware security vulnerabilities, secure processors were developed using cryptoprocessors, Trusted Execution Environment (TEE) and secure boot. Secure processors have a critical role in achieving high assurance of security systems. They provide a secure execution environment for guarding against unauthorised access to secret keys and decrypted information. Besides, such secure execution environment are critical to sensitive operations such as secure key generation, management and storage. Moreover, secure processors mitigate information leakage through main and side channels. In this article, we survey the state of the art literature on secure processors, by focusing our attention on protective measures for hardware, boot and the execution environment. We also present current trends and design challenges.**

***Keywords*—Hardware Security, Secure processor, TEE, Secure boot.**

I. Introduction

There is a growing need for securing wide areas of applications that we use in our daily life such as military, transportation, banking, finance sectors, IoT and many more. The need of data ciphering/cryptographic algorithm implementation in terms of software and hardware has escalated exponentially and different research groups from the world have worked out efficient implementation of cryptographic algorithms, primitives, and/or protocols in hardware which is called hardware cryptographic engines. In terms of flexibility this cryptographic engines can be classified as different types namely customized general-purpose processors, hardware cryptographic coprocessors (or hardware cryptographic accelerators), cryptoprocessors, and cryptographic coarse grained reconfigurable architecture (CGRA).

Many research groups have proposed various secure processors/cryptographic processor to provide hardware-level information security [1]. In various security modes [1], a system is partitioned into an on-chip trusted region and an off-chip untrusted region [2]. Physical attack resistance guarantees that the on-chip state of the processor (registers and caches) cannot be tampered with; though the off-chip component of the processor state may be observed and potentially modified by an adversary. Thus, they provide techniques to protect integrity and confidentiality of the off-chip memories, such as hash trees and memory encryption. Moreover, obfuscation techniques [3] are used in order to protect the hardware from being reverse engineered.

Scope and Contributions: In this paper, we have mainly focused on the diverse cryptoprocessors. Besides the crypto-

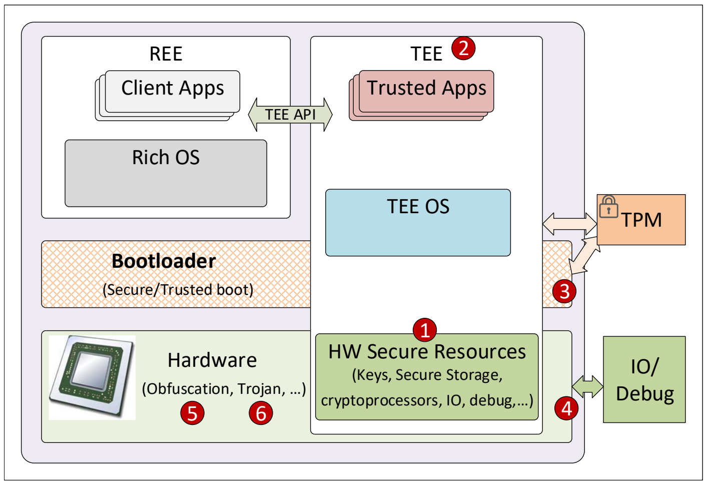

Fig. 1. Common structure of secure processors

processor, securing an embedded hardware sensitive information and for providing attack resistance, TEE and secure boot (Figure 1, 1-3) are also discussed in this article. Further design strategies, such as Hardware Security Module (HSM) and Address Space Layout Randomization (ASLR), as well as the brief discussion of Side Channel Attacks (SCA) are presented in this study as well. Other topics of secure processors, such as physical attacks resistance, hardware Obfuscation and hardware Trojans (Figure 1, 4-6), are not part of this article's scope. The contributions of this paper can be summarized as follows:

- Reviewing the state of the art literature on secure processors, by focusing our attention on protections for hardware, boot and the execution environment
- Providing recent comparison between representative cryptoprocessors and examining their characteristics. Additionally we highlighted various Trusted Execution Environments and their security and performance capabilities

The paper is organized as follows: Section II describes the different types of cryptoprocessors design overview followed by a comparison table. Section III discusses current TEE and its applications. In section IV, we introduce some background notions of secure boot technologies, both in academy and industry. Section V describes memory and physical side channel attack resistance on cryptographic/secure processor. A brief discussion on HSM is presented in Section III-A2 and ASLR is presented in Section V-C respectively. Finally, Section VI draws some conclusions.

II. Cryptoprocessors

Cryptoprocessor is a specialized processor that executes cryptographic algorithms on hardware level and accelerates the encryption and decryption process for better data security and secret key protection. There are many commercially available cryptoprocessors, such as IBM 4758, Maxim's DeepCover, SafeNet security processor, Atmel CryptoAuthentication devices, . Different cryptoprocessor designs [4] [5] [6] have been proposed to meet three essential criteria; security, flexibility and throughput. There are other constraints considered in designing cryptoprocessors, such as power consumption, area overhead, cost, dependability, etc. Here, we describe different cryptoprocessor architectures and its state of the art.

A. Customized General Purpose Processor

In the presented architecture, a general-purpose processor (GPP) is customized and used for implementing some cryptographic algorithms on it. It executes different functional modules for cryptographic algorithms, such as module exponentiation operation for RSA algorithm, AES S-Box operation, etc. The main customization of these processors are instruction set extensions (may be called cryptographic instruction) for cryptographic applications. In this case, secret keys are saved in the main memory and used like other normal data. Software attacks is rather easy on this type of architecture as it uses the same memory for storing the key and other data (e.g. cold boot attack [7]). Figure 2 shows a customized GPP for implementing cryptographic algorithms. CryptoBlaze [4] from

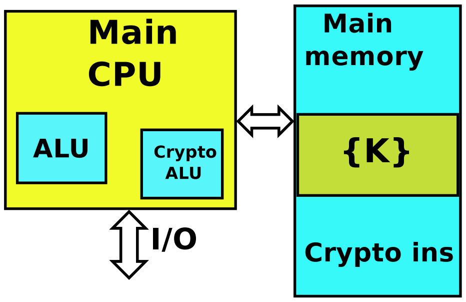

Fig. 2. Customized General Purpose Processor

Xilinx, the AES New Instructions (AES-NI) incorporated in the new Intel and AMD processors [8], [9] are examples of this kind of architecture.

B. Cryptographic coprocessor (crypto coprocessor)

A cryptographic coprocessor (also called cryptographic coprocessor) is a module outside the GPP that accelerates cryptographic computations. Here, cryptographic functions are implemented inside the coprocessor module in a very efficient way. Secret keys are normally not stored in the cryptographic coprocessor memory, they are stored as data in the processor data registers or main memory. The cryptographic coprocessor can be controlled or parametrised using the host GPP but it is not possible to modify/alter any instruction in the implementation flow of the coprocessor by the host GPP. These systems support multi-tasking, as the cryptographic functions run in the coprocessor, while the GPP executes other tasks simultaneously. A general multicore hardware coprocessor architecture design model is shown in Figure 3.

An AES coprocessor namely AESCrCU [10] is designed with a GPP and several reconfigurable Cryptographic Computation Units (CrCU) as shown in Figure 4. Secret keys

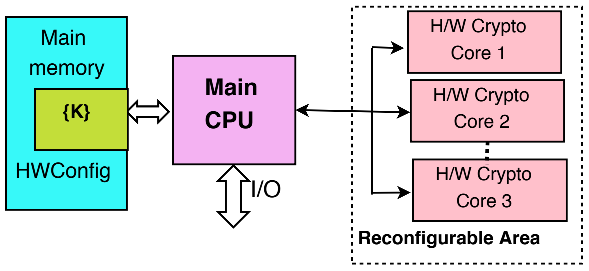

Fig. 3. Reconfigurable hardware cryptographic coprocessor

are stored in the Key register of the main memory which is connected to the AESCrCU coprocessor with standard bus technology.

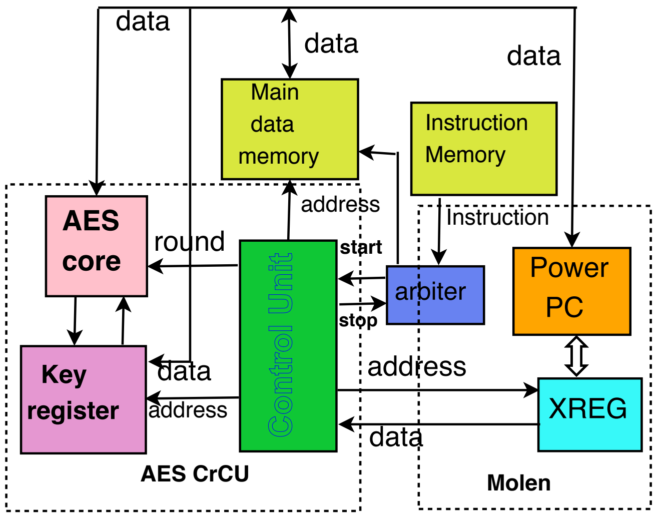

Fig. 4. AESCrCU coprocessor [10] architecture

Other example of cryptographic coprocessors are multicore AESTHETIC coprocessor [11], single core AES coprocessor [12] and dual AES core hardware implementation for AES MultiStream (AES-MS) [13], etc.

C. Cryptographic CGRA

In these type of design, different cryptographic algorithms are implemented using a CGRA [14] and are coupled with a GPP. Figure 5 shows a general cryptographic array architecture with GPP. Celator architecture [15] based AES state matrix

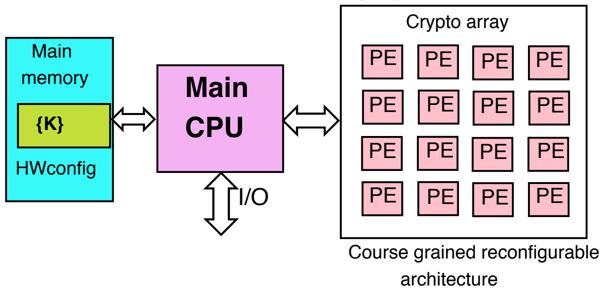

Fig. 5. GPP with cryptographic array

(AES-128) which is implemented with systolic array of 4×4, 8-bit processing elements (PE) is also a similar kind of architecture. Different symmetric cryptographic algorithms like DES, AES, SHA-256, etc. can also be implemented using Celator architecture. Interconnection of the Celator PE is configurable and a dedicated control logic is used to configure each Celator PE or the full architecture. Another cryptographic CGRA type architecture is COBRA array architecture [16], which is used for implementing cryptographic algorithm's basic function using 4×4, 32-bit RCEs (Reconfigurable Cryptographic Elements). Cryptographic functions like modular operations, shift/rotation operation, Galois Field Multiplication, Inversion, Substitution, etc. can be implemented using these RCE. Another new kind of implementation of AES alogorithm in Partially Reconfigurable (PR) CGRA called "reMORPH" is described in [17].

It is very hard and usually impossible to objectively compare performance of different cryptographic accelerator hardware implementation as different hardware setup, tools, boards, etc. are used for different experiments. Some results are given for the full system including communications overhead, etc. and some are only given for the cipher with/without taking care of key management process. That's why a straight comparison of these results may give a wrong judgment of that design. Results which are presented in Table I, are obtained from their respective published paper. There are large number of cryptographic accelerators and this table is representative of some of them which we have discussed in the Section II. This table represents different cryptoprocessor design characteristics likes key storage management, cryptographic units (single/multi core), performance (Mbps/MHz), etc. Interested readers are advised to read the respective paper for more details in regarding the experimental setup and analysis of the result.

III. Trusted Execution Environment

An execution environment is the software layer running on top of a hardware layer. A device that contains two execution environments that are physically separated; one environment contains the main OS and applications, the other environment contains trusted software components. The device has a physical separation between the trusted area and the untrusted area. The execution environment in the trusted area denoted as *Trusted Execution Environment (TEE)*, and the one in the untrusted area *Rich Execution Environment (REE)* are shown in Figure 6. The code executing in the TEE is guaranteed not to be tampered with as a function of the level of tamper-resistance exhibited by the hardware defining the trusted area that supports the TEE. A TEE can be realized in different ways, but the overall concept stays the same. Figure 7 shows a number of ways in which these TEEs can be realized:

A. Coprocessor based

As described in section II-B, the security coprocessor is a separate core, generally with its own peripherals, which is used to offload the security critical tasks from the main operating environment. The benefits of such a configuration are that the operation can generally be completely isolated and it can run simultaneously with the main core. The drawback is that there is an overhead associated with transferring the data to and from the core. Here we assumes that this communication channel in between coprocessor and external component is secure. The coprocessor generally has significantly lower performance than the main core. The coprocessor design can be one of two aternatives: *External* or *Embedded* security coprocessor.

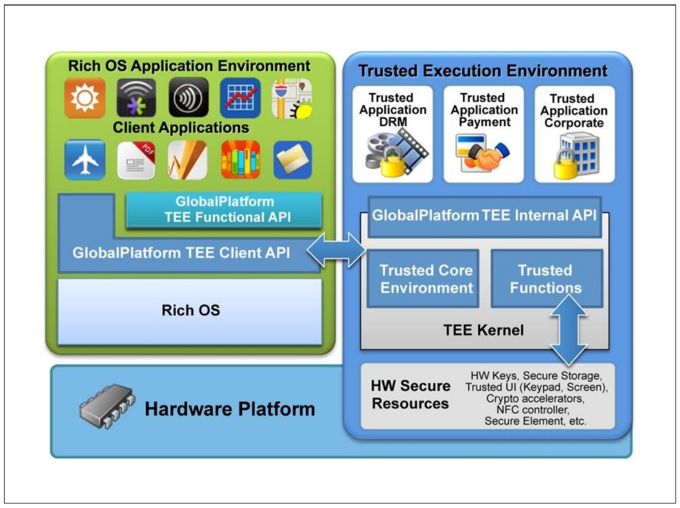
Rich OS Application Environment
Client Applications
GlobalPlatform TEE Functional API
GlobalPlatform TEE Client API
Rich OS
Trusted Execution Environment
Trusted Application DRM
Trusted Application Payment
Trusted Application Corporate
GlobalPlatform TEE Internal API
Trusted Core Environment
Trusted Functions
TEE Kernel
HW Secure Resources
HW Keys, Secure Storage, Trusted UI (Keypad, Screen), Crypto accelerators, NFC controller, Secure Element, etc.
Hardware Platform

Fig. 6. Trusted Execution Environment[20]

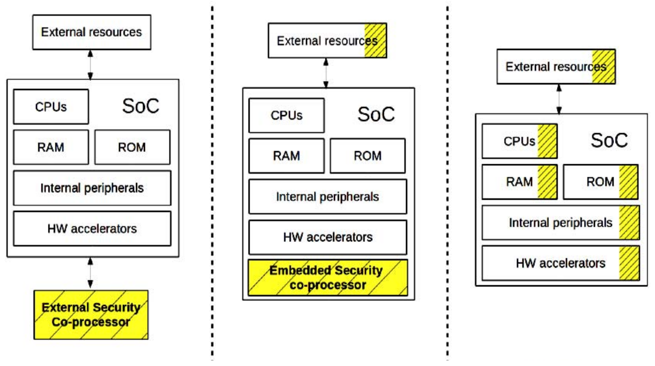

Fig. 7. Architectural options for realizing a TEE architecture [21]

1) *Trusted Platform Module:* The Trusted Platform Module (TPM) [22] is a specification for a secure cryptographic coprocessor defined by the Trusted Computing Group (TCG), and made an international standard (ISO/IEC 11889) in 2009. The TCG defines the TPM as a generator storage device and protector of symmetric keys [22], in general, it is used to store state, keys, passwords, and certificates. There are different types of TPMs implementations, at hardware level, there are the discrete TPMs (dTPM) and the integrated TPMs (iTPM). Discrete TPMs are chips that implement TPM functionality and nothing else, and are in their own semiconductor package. Integrated TPMs are part of another chip that implements other functionalities. Intel has integrated TPMs in some of its chipsets.

2) *Hardware Security Module:* A hardware security module (HSM) [23], [24] is a dedicated cryptographic processor that is specifically designed for the protection of the cryptographic key lifecycle. Hardware security modules act as trust anchors that protect the cryptographic infrastructure of computing systems by securely managing, processing, and storing cryptographic keys inside a hardened, tamper-resistant device. HSM traditionally come in the form of a plug-in card or an external device that attaches directly to a computer or network server. The dedicated cryptographic processor inside HSM provides performance accelerated cryptographic

TABLE I. DIFFERENT CRYPTOPROCESSOR DESIGN CHARACTERISTICS

Cryptoprocessor architecture	Proposed design name	Design architecture	Key storage management	Crypto core units	Mbps/MHz
Custom GPP	CryptoBlaze [4]	Custom 8-bit Xilinx PicoBlaze	Main memory	S-Box,GF multiplier on it	–
	Intel AES-NI [8], [9]	Intel IA-32	main memory	1 AES ALU	0.78
Crypto Coprocessor	AESTHETIC [11]	Main processor& AES accelerator	embedded key generation register	single & multiple AES core(1 to 3)	36.80 (3-core)
	AESCrCU [10]	Molen processor AES Cores &	key register	CrCU number not limited	0.59
	AES-MS [13]	Molen processor & 2 AES Cores	key register	two AES core	25.60
	HCrypt [18]	2×32 -bit dedicated ALU	dedicated secure register	2-AES cipher/decipher	1.60
	MCCP [19]	Multicore processor	dedicated register	2 to 8 reconfigurable crypto core	4.43
Crypto array	Celator arch. [15]	matrix of 4-bit PE	Main memory	Depends on array size	1.27
	COBRA array arch. [16]	4×4 32-bit RCE	Main memory	16-reconfigurable elements	1.40

operations such as encryption, digital signatures, hashing and Message Authentication Codes (MAC).

Generally, both HSM and TPM are capable of performing hardware encryption. A TPM is a hardware chip on the motherboard and is capable to provides full disks encryption. TPM keeps hard drives locked until the system completes a system verification or authentication process. On the other hand, HSM is a removable or external device that generates, stores and manages cryptographic keys. High performance HSM comes as an external device which can be connected to a network using TCP/IP while smaller HSM is expansion cards that is installed within a server or plug-in devices into computer ports.

B. Processor Secure Environment

Nowadays, many popular mobile TEE architectures support a new kind of configuration where a single core supports multiple virtual cores that are mutually exclusive of one another i.e. when one is running the other is suspended. Normally this transition from one state to other is done by some sort of monitor/trigger. This configuration is sometimes referred to as the "processor secure environment" [25].

1) ARM TrustZone: ARM TrustZone is an example of Processor Secure Environment. In Trust-Zone, the processor core can be in one of two "worlds": a "secure world" (for the TEE) and a "normal world" (for the REE). A specific instruction called Secure Monitor Call (SMC) is used to trigger the processor running in normal world to enter "monitor mode" that marshals the transition to secure world [26]. The benefit of this configuration is that there is no need to unload the data to and from the secure world. But there is an extra cost needed to store and restore the device state on entry and exit from a given mode. Disadvantage of this type of design is that when one world is active the other world must be completely halted, thus complicating interrupt handling.

2) Intel Trusted Execution Technology: Intel Trusted Execution Technology (Intel TXT, formerly known as LaGrande Technology) is a set of hardware security extensions to the Intel processors with the objective of providing a dynamic root of trust for measurement (DRTM), so that a system can be attested for integrity at run time. Intel TXT depends on the TPM's Platform Configuration Register (PCR) registers [22] to store its integrity measurements; thus the TPM is the root of trust for Intel TXT. One of the main applications of Intel TXT is ensuring a chain of trust at boot-time(e.g. Trusted boot).

C. Software Approaches

Beside the hardware based techniques, other approaches were presented, at the architecture level XOM (eXecute-Only-Memory), [27] platforms achieves TEE by architectural separation, it prevents information from leaking out of the XOM applications by using *compartment*, where a compartment is a logical container that prevents information from flowing into or out of it. Each application is placed on its compartment and implementing a policy that halts execution process if compartment tries to read data that its not part of its compartment boundaries.

Application level TEE techniques were presented at InkTag [28], XOM OS [27], and SP3 [29], these approaches share the goal of minimizing the ability of an untrustworthy system component to tamper with a sensitive application.

Additional examples of software TEE include Docker containers [30], Flicker [31] and Trusty [32] TEE for Android OS. Docker is a tool that makes it easy to package an application and all of its dependencies into such containers. Trusty is a set of software components supporting a TEE on mobile devices. Flicker demonstrates that it is possible to use current trusted computing and hardware virtualization technologies to implement TEE. With Flicker, current commodity systems are capable of securely executing code without the need to trust the legacy operating system.

D. Hybrid Software-Hardware Approaches

Techniques such as Intel Software Guard Extensions (Intel SGX)[33] uses both hardware and software to implement TEE. SGX is a set of instructions that allow an application to instantiate a protected container, referred to as an *enclave*.

An enclave is defined as a protected area in the application's address space which cannot be altered by code outside the enclave, not even by higher privileged code (e.g., kernel, virtual machine monitors, BIOS). In other words, enclaves are orthogonal to x86 protection rings(are mechanisms to protect data and functionality from faults (by improving fault tolerance) and malicious behavior (by providing computer security)) [34].

In SGX the core does not perform a full transition to and from a secure world. Instead parts of a standard application, both code and data, are protected by mechanisms in the core. The enclave is encrypted by a key that is only accessible to the CPU. When an "enter enclave (EENTER)" [35] instruction is received the code and data are decrypted and operated upon in the core. They never leave the CPU package unencrypted, thus protecting them against external access. The benefits are that there is no need to transfer data back and forth between cores or to setup complicated transitions to and from a secure world, and there is no additional need for a separate operating environment as is required in other styles of TEE configuration.

E. Summary of TEE

The desired security features [36] that TEE should provide are: Isolated Execution, Secure Storage, Remote Attestation, Secure Provisioning and Trusted Path, in addition the TEE system should maintain good performance (low overhead) and low power. Table II below compares part of the reviewed trusted execution environments based on these desired features.

IV. Security of Boot Process

Protecting the boot sequence is usually referred to indistinctly as *secure boot* or *trusted boot*. Other terms such as *verified boot*, *authenticated boot*, *certified boot*, or *measured boot* are also found in commercial products and some research articles, all with different connotations. In this section, we will define the terms according to their acceptance, and the relevance of the work referring to them.

A. Secure boot

The purpose of secure boot is to bring the system to a known and trusted state. The secure boot routine is a ROM based routine, so that an attacker cannot intercept the procedure. secure boot is described in the Unified Extensible Firmware Interface (UEFI) specification since version 2.2 [37]. UEFI secure boot verified the integrity of each stage of the boot process by computing a hash and comparing the result with a cryptographic signature. A key database of trustworthy public keys needs to be accessible during boot time so that the signature can be verified. In secure boot, if any integrity check fails, the boot will be aborted. If the boot process succeeds the system is expected to be running in a trusted state. This definition of secure boot is widely accepted in the security community [37], [38], [39].

B. Trusted Boot

Trusted Boot is defined by the TCG as part of the same set of standards [22] where they define the TPM. In this case, the TCG targets boot protection with trusted boot. The TCG describes a process to take integrity measurements of a system and store the result in a TPM so that they cannot be compromised by a malicious host system. In trusted boot, integrity measurements are stored for run-time attestation, but the boot process is not aborted in case of an integrity check failure. Also in mobile devices, where TPMs are not available, the TCG and others are working on materializing a TPM-like hardware component called Mobile Trusted Module (MTM) [40]. They are not being very successful in this effort due to the lack of tamper-resistance in TrustZone, which is the most widespread secure hardware in mobile devices. Since this definition of trusted boot, originally introduced by Gasser et al. as part of their digital distributed system security architecture [41], is backed by the TCG, its acceptance is also widely spread. Also, most of the community has adhered to it [39], [42]. Popular trusted boot implementations include OSLO [43], TCG's Software Stack (TrouSerS), and tboot.

C. Certified Boot

Certified Boot has been used in [37]as a term to combine secure boot and trusted boot in such a way that integrity checks are both stored and enforced. Since the introduction of authorizations in TCG's TPM 2.0 [44], certified boot is also defined as secure boot with TCG authorizations, with the latter referring to trusted boot.

D. Measured Boot

Measured Boot is a mechanism to verify that core platform software components have not been compromised. Measured boot starts after certified boot. Typically, measured boot is supported by a TPM's PCR registers [45]. Conceptually, it relies on the idea that each running core code component is measured (âĂIJmeasuresâĂİ or computes the hash of, the next object(s) in the chain of trust, and stores the hashes in a way that they can be securely retrieved later to find out what objects were encountered) before it is executed. Several works adhere to this definition when using a TPM for storing the measurements in the PCR registers [46], [47].

E. Verified Boot

Verified Boot is a term coined by Google in Chrome OS for verifying that the firmware and the file system that holds the operating system are in a known, untampered state. Conceptually, verified boot is very similar to trusted boot. However, instead of using hash values to perform integrity checks, they use only digital signatures. Work targeting Chrome OS have already adopted this term [48]. Also, given Google products' popularity, at least two independent efforts are driving its development: one by Google's Chromium OS team, and another inside of U-Boot [48], [49]. The fact that verified boot is directly supported in U-Boot guarantees it reaching a large proportion of the available target platforms.

F. Authenticated Boot

Authenticated Boot is sometimes used to designate trusted boot [50], [51]. The logic behind this is that trusted boot can then be used to define the combination of secure boot and authenticated boot, in such a way that integrity checks are both stored and enforced. This is what we referred to before as certified boot.

V. Memory and Cryptographic Attack Resistance

Several prominent attacks (using side channels) that imposed a significant challenge in secure processor designs are described in the following subsections. While cryptographic algorithms implementations are commonly deployed to countermeasure against invasions, other measures such as the ASLR and Oblivious RAM, which could increase the resistance against memory attack are also introduced in this study.

A. Side Channel Attacks (Non Invasive Attack)

Side Channel Attacks (SCA) are a low-cost and non-invasive physical attack that exploits device's external information which is often leaked unintentionally in exchange for secret information. Several commonly deployed SCA use the analysis of execution time, power consumption and electromagnetic radiation are introduced in this section.

1) Timing Analysis Attack: Operations in cryptographic algorithms require different computation time to process different inputs. Therefore, with a sufficient knowledge of the implementation, a careful statistical analysis of the variations on timing measurement will eventually lead to private information, i.e. (part of) the secret key. Timing analysis attack was first introduced by Kocher in 1996 [52], and was developed by Dhem et. al.[53], in which a practical timing attack was conducted against RSA modular exponentiation. A complete AES key recovery from a network server on another computer using timing attack was demonstrated in [54].

TABLE II. Comparison between different TEE approaches

	Performance	Isolated Execution	Secure Storage	Remote Attestation	Secure Provisioning	Trusted Path
TPM –Hardware	much lower than main CPU	Limited	Yes (limited capacity)	Yes	Yes	No
TrustZone -Hardware	near CPU, RAM limited	Yes (restricted)	Yes	Yes	Yes	Partially
Flicker -Software	as main CPU	Yes (restricted)	Yes (via TPM)	Yes (via TPM)	Yes (via TPM)	Limited
SGX -Hybrid	as main CPU	Yes	Yes	Yes	Yes	Yes

2) Power Analysis Attack: Apart from the computation runtime, the power consumption of a cryptographic devices also reveals much information about the operation taking place as well as the underlying parameters. Hence, secret data can be retrieved through power analysis attacks such as Simple Power Analysis (SPA), Differential Power Analysis (DPA) and Correlation Power Analysis (CPA).

SPA is a technique that involves directly interpreting single power trace to determine the particular operation executed and hence leads to the key materials [55]. However, when the leakages relative to noise is much less, statistical techniques such as DPA are more applicable. DPA, on the other hand, performs statistical analysis over a large number of samples. The approach was successfully demonstrated on DES algorithm by Kocher et al.[56], [57]. This statistical method exploited the signal's power consumption signals on CMOS chips in order to retrieve the key values via the difference of mean curves selected on a defined criteria.

In recent dates, the power analysis technique based on the correlation factor between the power traces and the processing data has been widely studied. This approach, known as CPA, exploits the correlation between the power consumption of a cryptographic device and its power consumption model [58]. The most common model given under a linear form is such as the Hamming weight model and the Hamming distance model. Power traces recorded during the encryption process are correlated with a table of expected power consumption and the process repeated until accurate inference can be made about the key [59], [60]. CPA requires far fewer traces for recovering the key and hence is superior than methods such as DPA and SPA.

Apart from power dissipation, the analysis of the observed electromagnetic (EM) emanations could also reveal the causal relationship between the processing data and the underlying computation as well. Hereafter, the terms DPA and CPA have been generalized for any side channel signal (i.e., power consumption and electromagnetic radiation signals) [61], [62].

3) Template Attack: The effectiveness of DPA and CPA attacks are dependent on the selected power consumption models. If the power consumption is wrongly modeled, the key detection will become impossible. With that, template attack, a new class of power attacks was introduced by Chari et. al. to overcome these shortcomings [63]. The approach consists of two stages: profiling stage to learn the device and key extraction stage to retrieve the secret key [64], [63]. The power consumption characteristics of the targeted device is learnt and stored as templates in a database. The templates are later matched with the recorded power traces based on different key hypotheses. The templates that match best indicate the key.

B. Fault Analysis Attack (Semi Invasive Attack)

Fault analysis attacks are semi invasive attack that attempt to access the device without damaging the passivation layer. The attack generally requires two step: fault injection and fault exploitation steps. Faults are first injected at the appropriate timing during the execution process. These faults can come in the forms of abruptly low or high voltage, noisy power or clock signals, excessive temperature, radiations and high energy beam such as UV or laser. Next, by carefully studying the erroneous results of computations performed under such faults, an attacker can retrieve information about the secret key. Fault analysis attack was first introduced by Boneh et. al. and was demonstrated on RSA public key cryptosystem [65]. Subsequently, the approach was extended to symmetric key cryptosystems, among others, DES [66] and AES [67].

C. Address Space Layout Randomization

Another major concern with respect to existing different security attacks, is memory attack. Any change in the instruction memory/data memory content of the processor, will change the flow of execution or chances to hack the secret key/data stored in that memory location. Different memory protection techniques are used to countermeasure of this kind of attacks. ASLR is a memory protection approach that is used to fortify operating systems against buffer-overflow attacks [68], [69]. The technique works by introducing artificial diversity through randomizing the memory location of certain system components. The aim is to prevent attackers from utilizing the same attack code against all instantiations of the program containing the same flaw. In other words, ASLR hinders the attackers to predict the randomly assigned addresses. For example, attackers are required to locate the targeted stack before executing the *return-to-libc* attacks. However, it has been proven that for current 32-bit architectures, ASLR is ineffective against the possibility of generic exploit code for single flaw and brute force attacks can be effectively feasible as well [70]. Furthermore, de-randomization attack converts standard buffer-overflow exploits into exploits that works against systems protected by ASLR [71]. Several new and improved ASLR architectures have been investigated and reported in the recent years. These works suggested upgrading to 64-bit machine with no fewer than 32 address bits [70]. Brute force attacks would then need to predict at least 32 bits of randomness and this can be ruled out as a threat.

In addition to that, more powerful randomization techniques such as increasing the frequency and granularity of the randomization can be implemented [68]. Randomization of the processes' memory segments can be performed to increase the security against brute force attacks. The approach can be extended to the finer granularity level (at function and variable addresses) within the memory segments during compile-time or run-time. Compile-time randomization can be achieved via modified compiler and linker [72]. Meanwhile, run-time randomization is performed through function reordering within a shared library or the executable during the runtime [68]. In comparison, the compile-time address space randomization is more effective than runtime randomization because it has higher feasibility to randomize addresses at a finer granularity.

D. Oblivious RAM

Oblivious RAM (ORAM) [73], is a cryptographic primitive that completely eliminates the information leakage from a program memory access trace, i.e. the sequence of memory accesses. ORAM is made up of trusted client logic (who runs the ORAM algorithm and maintains some trusted state) that interacts with an untrusted storage provider. Under ORAM, any memory access pattern is computationally indistinguishable from any other access pattern of the same length. ORAM requires flexible amounts of trusted memory so that the attacker cannot observe the memory access pattern. Most secure processor designs [74] [75] have adopted Path ORAM due to its algorithmic simplicity.

VI. Conclusion

With the rapid increase in utilization of hardware circuits in larger CPS, many new security threats associated with the system hardware have emerged. In this article, we have surveyed cryptoprocessors and their applications. In addition, we showed the hardware related vulnerabilities and countermeasures that makes hardware systems more secure and reliable. Various cryptoprocessors were surveyed. In addition, we have presented the various TEE methods, and the multiple approaches for secure boot. It can be observed that, trusted boot, trusted execution-environment, secure debug, countermeasure and secured storage are the main features that are required for securing the hardware and the software that is running on it.

References

[1] R. Kannavara and N. G. Bourbakis, "Surveying secure processors," *IEEE Potentials*, vol. 28, no. 1, pp. 28–34, 2009.

[2] G. E. Suh, D. Clarke, B. Gassend, M. Van Dijk, and S. Devadas, "Aegis: architecture for tamper-evident and tamper-resistant processing," in *Proceedings of the 17th annual international conference on Supercomputing*, pp. 160–171, ACM, 2003.

[3] X. Zhuang, T. Zhang, H.-H. S. Lee, and S. Pande, "Hardware assisted control flow obfuscation for embedded processors," in *Proceedings of the 2004 international conference on Compilers, architecture, and synthesis for embedded systems*, pp. 292–302, ACM, 2004.

[4] A. Note and CPLD, "Cryptoblaze: 8-bit security microcontroller," *CoolRunner-II*, 2003.

[5] A. Hodjat and I. Verbauwhede, "Interfacing a high speed crypto accelerator to an embedded CPU," in *Signals, Systems and Computers, 2004. Conference Record of the Thirty-Eighth Asilomar Conference on*, vol. 1, pp. 488–492, IEEE, 2004.

[6] C.-P. Su, C.-L. Horng, C.-T. Huang, and C.-W. Wu, "A configurable AES processor for enhanced security," in *Proceedings of the 2005 Asia and South Pacific Design Automation Conference*, ASP-DAC '05, (New York, NY, USA), pp. 361–366, ACM, 2005.

[7] 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," *Communications of the ACM*, vol. 52, no. 5, pp. 91–98, 2009.

[8] S. Bartolini, R. Giorgi, and E. Martinelli, "Instruction set extensions for cryptographic applications," in *Cryptographic Engineering*, pp. 191–233, Springer, 2009.

[9] S. Gueron, "Intel® advanced encryption standard (AES) new instructions set," *Intel Corporation*, 2010.

[10] A. Hodjat, "Reconfigurable cryptographic processor," in *InProceedings Of The Workshop On Circuits, Systems And Signal Processing (Prorisc 2006)*, IEEE Press, 2006.

[11] M.-Y. Wang, C.-P. Su, C.-L. Horng, C.-W. Wu, and C.-T. Huang, "Single- and multi-core configurable AES architectures for flexible security," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 18, pp. 541–552, Apr. 2010.

[12] A. Hodjat and I. Verbauwhede, "High-throughput programmable cryptocoprocessor," *IEEE Micro*, vol. 24, pp. 34–45, May 2004.

[13] M. Pericas, R. Chaves, G. N. Gaydadjiev, S. Vassiliadis, and M. Valero, "Vectorized AES core for high-throughput secure environments," in *International Conference on High Performance Computing for Computational Science*, pp. 83–94, Springer, 2008.

[14] K. Shahzad, A. Khalid, Z. E. Rakossy, G. Paul, and A. Chattopadhyay, "Coarx: A coprocessor for arx-based cryptographic algorithms," in *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–10, May 2013.

[15] D. Fronte, A. Perez, and E. Payrat, "Celator: A multi-algorithm cryptographic co-processor," in *Proceedings of the 2008 International Conference on Reconfigurable Computing and FPGAs*, RECONFIG '08, (Washington, DC, USA), pp. 438–443, IEEE Computer Society, 2008.

[16] A. J. Elbirt and C. Paar, "Instruction-level distributed processing for symmetric-key cryptography," in *Proceedings International Parallel and Distributed Processing Symposium*, pp. 10 pp.–, April 2003.

[17] K. Paul, C. Dash, and M. S. Moghaddam, "remorph: A runtime reconfigurable architecture," in *2012 15th Euromicro Conference on Digital System Design*, pp. 26–33, Sept 2012.

[18] L. Gaspar, V. Fischer, F. Bernard, L. Bossuet, and P. Cotret, "Hcrypt: A novel concept of crypto-processor with secured key management," in *2010 International Conference on Reconfigurable Computing and FPGAs*, pp. 280–285, Dec 2010.

[19] M. Grand, L. Bossuet, B. Le Gal, G. Gogniat, and D. Dallet, "Design and implementation of a multi-core crypto-processor for software defined radios," in *Reconfigurable Computing: Architectures, Tools and Applications: 7th International Symposium, ARC 2011, Belfast, UK, March 23-25, 2011. Proceedings* (A. Koch, R. Krishnamurthy, J. McAllister, R. Woods, and T. El-Ghazawi, eds.), pp. 29–40, Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.

[20] G. Platform, "The trusted execution environment: Delivering enhanced security at a lower cost to the mobile market," *White Paper February*, 2011.

[21] "Tee system architecture." [Online]. Available: http://www.globalplatform.org/specificationsdevice.asp.

[22] T. C. Group, "TPM main specification level 2 version 1.2, revision 116," 2011.

[23] S. Smith, "Hardware security modules," in *Handbook of Financial Cryptography and Security* (B. Rosenberg, ed.), pp. 257–278, Chapman and Hall/CRC, 2010.

[24] J. Pescatore, "Using hardware-enabled trusted crypto to thwart advanced threats." [SANS Institute InfoSec Reading Room]. Available: https://www.sans.org/reading-room/whitepapers/analyst/hardware-enabled-trusted-crypto-thwart-advanced-threats-36215, 2015.

[25] J. Ekberg, "Securing software architectures for trusted processor environments." [Doctoral dissertation, Aalto University, May 2013]. Available: http://urn.fi/URN:ISBN:978-952-60-3632-8.

[26] ARM, "Arm security technology âĂŤ building a secure system using trustzone technology." [Online]. Available: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.prd29-genc-009492c/index.html,April2009.

[27] D. Lie, C. A. Thekkath, and M. Horowitz, "Implementing an untrusted operating system on trusted hardware," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 178–192, 2003.

[28] O. S. Hofmann, S. Kim, A. M. Dunn, M. Z. Lee, and E. Witchel, "Inktag: Secure applications on an untrusted operating system," in *ACM SIGARCH Computer Architecture News*, vol. 41(1), pp. 265–278, ACM, 2013.

[29] J. Yang and K. G. Shin, "Using hypervisor to provide data secrecy for user applications on a per-page basis," in *Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pp. 71–80, ACM, 2008.

[30] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux Journal*, vol. 2014, no. 239, p. 2, 2014.

[31] J. M. McCune, B. J. Parno, A. Perrig, M. K. Reiter, and H. Isozaki, "Flicker: An execution infrastructure for tcb minimization," in *ACM*

SIGOPS Operating Systems Review, vol. 42(4), pp. 315–328, ACM, 2008.

[32] Android, "The android source code: Trusty tee," *Android Open Source Project*, 2016.

[33] I. Anati, F. Mckeen, S. Gueron, H. Haitao, S. Johnson, R. Leslie-Hurd, H. Patil, C. Rozas, and H. Shafi, "Intel software guard extensions (Intel sgx)," in *Tutorial at International Symposium on Computer Architecture (ISCA). IEEE Computer Society*, 2015.

[34] Intel, "Intel-64 and IA-32 architectures software developer's manual," *Volume 3A: System Programming Guide, Part*, vol. 1, no. 64, 2014.

[35] F. McKeen, I. Alexandrovich, I. Anati, D. Caspi, S. Johnson, R. Leslie-Hurd, and C. Rozas, "Intel software guard extensions (Intel SGX) support for dynamic memory management inside an enclave," in *Proceedings of the Hardware and Architectural Support for Security and Privacy 2016*, p. 10, ACM, 2016.

[36] A. Vasudevan, E. Owusu, Z. Zhou, J. Newsome, and J. M. McCune, "Trustworthy execution on mobile devices: What security properties can my mobile platform give me?," in *International Conference on Trust and Trustworthy Computing*, pp. 159–178, Springer, 2012.

[37] I. U. EFI, "Unified extensible firmware interface specification: Version 2.2 d," 2010.

[38] S. Mukkamala, G. Janoski, and A. Sung, "Intrusion detection using neural networks and support vector machines," in *Neural Networks, 2002. IJCNN'02. Proceedings of the 2002 International Joint Conference on*, vol. 2, pp. 1702–1707, IEEE, 2002.

[39] R. Sailer, X. Zhang, T. Jaeger, and L. Van Doorn, "Design and implementation of a TCG-based integrity measurement architecture.," in *USENIX Security Symposium*, vol. 13, pp. 223–238, 2004.

[40] T. M. P. W. Group *et al.*, "TPM mobile with trusted execution environment for comprehensive mobile device security," *Trusted Computing Group*, 2010.

[41] M. Gasser, A. Goldstein, C. Kaufman, and B. Lampson, "The digital distributed system security architecture," in *Proceedings of the 12th National Computer Security Conference*, pp. 305–319, 1989.

[42] B. J. Parno, *Trust Extension As a Mechanism for Secure Code Execution on Commodity Computers*. New York, NY, USA: Association for Computing Machinery and Morgan &#38; Claypool, 2014.

[43] B. Kauer, "OSLO: Improving the security of trusted computing.," in *USENIX Security*, vol. 7, 2007.

[44] T. C. Group, "TPM 2.0 main specification family 2.0, level 00, revision 01.16.," -, 2014.

[45] TCG, "Client specific implementation specification for conventional bios," *Specification Version*, vol. 1, 2012.

[46] V. Zimmer, M. Rothman, and S. Marisetty, *Beyond BIOS: developing with the unified extensible firmware interface*. Intel Press, 2010.

[47] S. Pearson and B. Balacheff, *Trusted computing platforms: TCPA technology in context*. Prentice Hall Professional, 2003.

[48] S. Glass., "Verified u-boot." http://lwn.net/Articles/571031/. October 2013.

[49] S. Glass, "Verified boot on chrome os and how to do it yourself," in *Embedded Linux Conference Europe*, (660 York Street, Suite 102, San Francisco, CA 94110, USA), 2013.

[50] H. Löhr, A.-R. Sadeghi, C. Stüble, M. Weber, and M. Winandy, "Modeling trusted computing support in a protection profile for high assurance security kernels," in *International Conference on Trusted Computing*, pp. 45–62, Springer, 2009.

[51] W. A. Arbaugh, "Trusted computing," *Department of Computer Science, University of Maryland*, [online]Retrieved on Feb. 22, 2007.

[52] P. C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA,DSS, and other systems," in *Advances in Cryptology — CRYPTO '96: 16th Annual International Cryptology Conference Santa Barbara, California, USA August 18–22, 1996 Proceedings* (N. Koblitz, ed.), pp. 104–113, Berlin, Heidelberg: Springer Berlin Heidelberg, 1996.

[53] J. F. Dhem, F. Koeune, P. A. Leroux, P. Mestre, J. J. Quisquater, and J. L. Willems, "A practical implementation of the timing attack," in *Smart Card Research and Applications: Third International Conference, CARDIS'98, Louvain-la-Neuve, Belgium, September 14-16, 1998. Proceedings* (J.-J. Quisquater and B. Schneier, eds.), pp. 167–182, Berlin, Heidelberg: Springer Berlin Heidelberg, 2000.

[54] J. B. Daniel, "Cache-timing attacks on AES," tech. rep., 2005.

[55] P. Kocher, J. Jaffe, and P. Jun, B. Rohatgi, "Introduction to differential power analysis," *Journal of Cryptographic Engineering*, vol. 1, no. 1, pp. 5–27, 2011.

[56] P. Kocher, J. Jaffe, and B. Jun, "Introduction to differential power analysis and related attacks," 1998. URL: www.cryptography.com/resources/whitepapers/DPATechInfo.pdf.

[57] P. Kocher, J. Jaffe, and B. Jun, *Differential Power Analysis*, pp. 388–397. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999.

[58] E. Brier, C. Clavier, and F. Olivier, *Correlation Power Analysis with a Leakage Model*, pp. 16–29. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004.

[59] T. H. Le, J. Clédière, C. Canovas, B. Robisson, C. Servière, and J. L. Lacoume, "A proposition for correlation power analysis enhancement," in *Proceedings of the 8th International Conference on Cryptographic Hardware and Embedded Systems*, CHES'06, (Berlin, Heidelberg), pp. 174–186, Springer-Verlag, 2006.

[60] N. Benhadjyoussef, H. Mestiri, M. Machhout, and R. Tourki, "Implementation of CPA analysis against AES design on FPGA," in *2012 International Conference on Communications and Information Technology (ICCIT)*, pp. 124–128, June 2012.

[61] K. Gandolfi, C. Mourtel, and F. Olivier, *Electromagnetic Analysis: Concrete Results*, pp. 251–261. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001.

[62] R. R. Josyula and R. Pankaj, "EMpowering side-channel attacks," 2001.

[63] S. Chari, J. R. Rao, and P. Rohatgi, *Template Attacks*, pp. 13–28. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003.

[64] P. N. Fahn and P. K. Pearson, *IPA: A New Class of Power Attacks*, pp. 173–186. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999.

[65] D. Boneh, R. A. DeMillo, and R. J. Lipton, *On the Importance of Checking Cryptographic Protocols for Faults*, pp. 37–51. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997.

[66] E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems," in *Proceedings of the 17th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '97, (London, UK, UK), pp. 513–525, Springer-Verlag, 1997.

[67] P. Dusart, G. Letourneux, and O. Vivolo, *Differential Fault Analysis on A.E.S*, pp. 293–306. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003.

[68] H. Shacham, M. Page, B. Pfaff, E.-J. Goh, N. Modadugu, and D. Boneh, "On the effectiveness of address-space randomization," in *Proceedings of the 11th ACM Conference on Computer and Communications Security*, CCS '04, (New York, NY, USA), pp. 298–307, ACM, 2004.

[69] B. Gras, K. Razavi, E. Bosman, H. Bos, and C. Giuffrida, "ASLR on the Line: Practical Cache Attacks on the MMU," in *NDSS*, Feb. 2017.

[70] J. S. Chase, H. M. Levy, M. Baker-Harvey, and E. D. Lazowska, "How to use a 64-bit virtual address space," in *Department of Computer Science and Engineering, University of Washington*, Citeseer, 1992.

[71] T. Durden, "Bypassing pax aslr protection." [Phrack Magazine]. Available: http://www.phrack.org/archives/59/p59-0x09, 2002.

[72] E. Bhatkar, D. C. Duvarney, and R. Sekar, "Address obfuscation: an efficient approach to combat a broad range of memory error exploits," in *In Proceedings of the 12th USENIX Security Symposium*, pp. 105–120, 2003.

[73] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious RAMs," *J. ACM*, vol. 43, pp. 431–473, May 1996.

[74] L. Ren, X. Yu, C. W. Fletcher, M. van Dijk, and S. Devadas, "Design space exploration and optimization of path oblivious RAM in secure processors," *SIGARCH Comput. Archit. News*, vol. 41, pp. 571–582, June 2013.

[75] 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 &#38; Communications Security*, CCS '13, (New York, NY, USA), pp. 299–310, ACM, 2013.