A Survey on RISC-V Security: Hardware and Architecture

TAO LU*, Marvell Semiconductor Ltd., USA

The Internet of Things (IoT) is an ongoing technological revolution. Embedded processors are the processing engines of smart IoT devices. For decades, these processors were mainly based on the Arm instruction set architecture (ISA). In recent years, the free and open RISC-V ISA standard has attracted the attention of industry and academia and is becoming the mainstream. Many companies have already owned or are designing RISC-V processors. Many important operating systems and major tool chains have supported RISC-V. Data security and user privacy protection are common challenges faced by all IoT devices. In order to deal with foreseeable security threats, the RISC-V community is studying security solutions aimed at achieving a root of trust (RoT) and ensuring that sensitive information on RISC-V devices is not tampered with or leaked. Many RISC-V security research projects are underway, but the academic community has not yet conducted a comprehensive survey of RISC-V security solutions. The latest technology and future development direction of RICS-V security research are still unclear. In order to fill this research gap, this paper presents an in-depth survey on RISC-V security technologies. This paper summarizes the representative security mechanisms of RISC-V hardware and architecture. Specifically, we first briefly introduce the background and development status of RISC-V, and compare the RISC-V mechanisms with the most relevant Arm mechanisms, highlighting their similarities and differences. Then, we investigate the security research of RISC-V around the theme of hardware and architecture security. Our survey covers hardware and physical access security, hardware-assisted security units, ISA security extensions, memory protection, cryptographic primitives, and side-channel attack protection. Based on our survey, we predict the future research and development directions of RISC-V security. We hope that our research can inspire RISC-V researchers and developers.

CCS Concepts: • **Security and privacy** → **Embedded systems security**; **Cryptography**.

1 INTRODUCTION

The instruction set architecture (ISA) is an abstract model of a computer. ISA specifies the behavior of the running machine code without relying on specific machine implementations, thereby providing program compatibility between different implementations of the same architecture. The architecture specification conceptually defines the basic interface between hardware and software, the behaviors allowed for processor implementation, and the basic assumptions for software development and verification [1]. Inspired by proprietary ISA's IP restrictions together with the lack of 64-bit addresses and overall complexity [2], RISC-V was developed and became more and more popular. RISC-V aims to become a standard and universal ISA, especially for three representative device categories: small IoT devices, personal mobile devices, and warehouse-level computers.

After years of technological evolution, RISC-V has become a commercially available ISA. Currently, semiconductor companies are testing RISC-V, and even some product lines are transitioning to RISC-V. From the Google Scholar statistics of the RISC-V research work, we observe that the volume of RISC-V related research has exponentially increased in the past decade. The tremendous momentum of RISC-V adoption in computing platforms is very clear. Various RISC-V devices from small IoT microcontrollers to multi-core high-performance processors have been taped out. Recently, SiFive has taped out *HiFive1 Rev B* and *HiFive Unmatched* SoCs [3], which have been put on market for IoT and desktop applications. SiFive RISC-V cores have already been used for SSD Controllers [4]. Alibaba has taped out *Xuantie-910* for cloud and edge computing [5]. Other tape-outs and FPGA boards include Microchip PolarFire SoC FPGA Icicle Kit, RISC-V multicore accelerator SoC BlackParrot [6], Xilinx multi-core FPGA system VC707 [7] etc. Lee et al. [8] implemented a metamorphic IoT platform for on-demand hardware replacement in large-scaled IoT application scenarios. All these post-silicon implementations push RISC-V from concepts to products. European Processor Initiative (EPI), one of the cornerstones of the EuroHPC Joint Undertaking, a new European Union strategic entity focused on pooling the Union's and national resources on HPC to build and deploy the most powerful supercomputers within Europe, is preparing to adopt RISC-V as its core solution for exascale embedded HPC platform [9].

*This research is fully self-sponsored by the author. It does not represent the views of Marvell Semiconductor Ltd.

Author's address: Tao Lu, taolyu6@gmail.com, Marvell Semiconductor Ltd., USA.

Table 1 lists existing post-silicon RISC-V chips, which are mainly used in smart devices and Internet of Things (IoT) devices. IoT consists of billions of connected devices, is changing fields such as medical care, transportation, and public services, and continues to collect, process, and transmit big data. IDC predicts that there will be 41.6 billion connected IoT devices by 2025, and the data generated will exceed 79ZB [10]. The value of big data is widely recognized by the industry. The effective mining of big data can improve the competitive advantage of enterprises and provide a basis for the decision-making of social functional departments. However, the collection, storage, analysis and sharing of big data has brought new information security and privacy issues. Security has become a prominent challenge in the era of big data [11–13]. We expect that the security mechanism of the RISC-V architecture will play an important role in its future ecosystem.

With the development of cryptographic technology, many security mechanisms have been widely deployed in practice, including data confidentiality and integrity protection, identity verification, privacy protection, denial of service prevention, non-repudiation enforcement, and digital content protection. Various security protocols and standards such as TLS, ZRTP, IPSec, IKE, and Kerberos have been used to protect data services and applications and alleviate platform security challenges. However, with technology advancement, the sophistication of attacks is developing simultaneously, especially cyber attacks are more complicated, illusory and more targeted than ever [14]. In September 2019, iPhone hackers were exposed. For at least two years, attackers have used infected websites to Exploit 14 independent vulnerabilities in Apple iOS and install spyware on thousands of Apple devices that have visited websites infected with malware. Attackers can access regular user data, keychain passwords, and social media content [15]. Recently, FireEye, a publicly-listed cyber security company, was attacked by a highly sophisticated adversary who stole FireEye Red Team tools, which may be used for malicious cyber attacks on the system [16]. FireEye was forced to publish hundreds of countermeasures, So that the wider security community can protect itself from these tools. It is clear from the incident that no organization, whether a sophisticated security defender or not, is immune to destructive cyber attacks. According to a cyber security report released by Security Boulevard [17], cybercrime caused approximately $1.5 trillion in losses for victims in 2018. AI-driven attacks [18, 19] make the situation worse and make security defenses an increasingly serious challenge. In addition to countless cyber attacks, the disclosures of Meltdown [20] and Spectre [21] also revealed hardware vulnerabilities in modern processors. The Spectre and Meltdown attacks confirmed the need to treat security as a system-level design constraint that crosses the boundaries of hardware and software. The serious impact of ISA vulnerabilities has aroused unprecedented attention in the industry to architecture security. Side channel attacks have become widely known and have attracted a lot of research.

The market no longer only focuses on product performance, security is also a demanding requirement. At the application layer, security mechanisms for artificial intelligence, Internet of Things, and wireless sensor network platforms are proposed [22–33]. At the system level, security mechanisms such as trusted boot [34–39] and trusted execution [40–49] are widely used. IoT systems usually run on small cores of embedded systems. These cores have low computing power and limited resources, making it difficult to implement complicated security policies. Due to the high efficiency of hardware execution, hardware-based security mechanisms can minimize the resource cost of these devices. Arm TrustZone, Intel SGX, and AMD SEV technologies provide system-wide security solutions through hardware isolation implemented by the CPU [50]. Although the existing hardware isolation technology is not impeccable, for example, the TEE system assisted by TrustZone has security vulnerabilities [51], it still plays an important role. Enhancing hardware and architecture security is an important requirement. Balancing platform security level and system performance, hardware and architecture security is important for platform security solutions. Therefore, chip suppliers are now actively introducing hardware security modules (HSM) and are very careful to avoid security vulnerabilities in hardware design.

The openness of RISC-V enables public auditing of the architecture design, thereby providing opportunities to build secure platforms. However, the openness of ISA provides attackers with more details behind the scenes, and system security vulnerabilities can be more easily discovered and exploited by adversaries. Therefore, RISC-V needs to use its openness to build reliable security mechanisms. The RISC-V security community needs to understand this relatively new architecture to carry out security technology innovations. RISC-V supports various privilege modes [52] and physical memory protection [52–54]. Trusted execution environments [40–49] have also been implemented. Other security enhancement measures including hardware security [55–68], memory

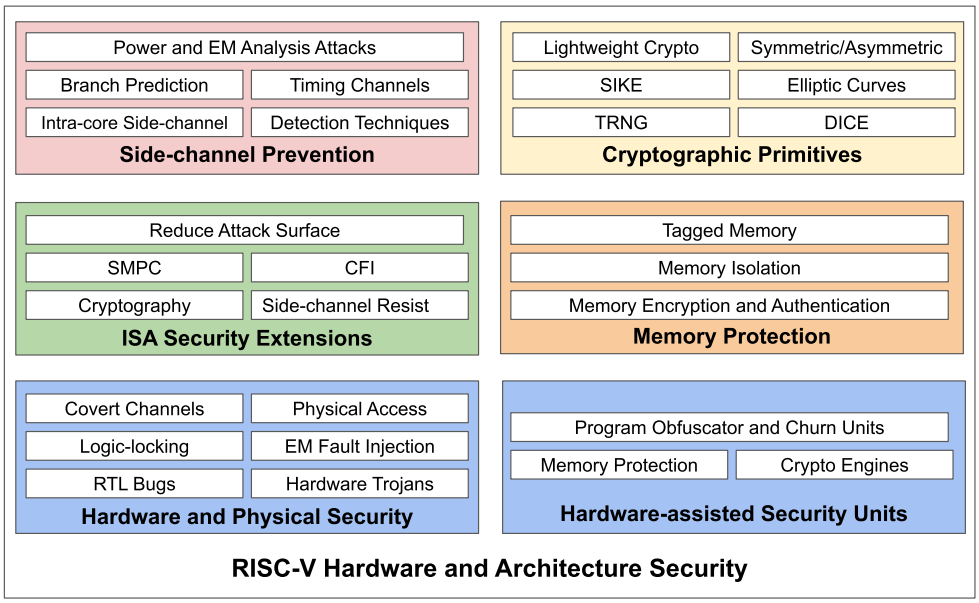


Fig. 1. The spectrum of RISC-V hardware and architecture security, which is the cornerstone of platform security.

protection [54, 56, 57, 59, 61, 69–72], ISA security extensions [66, 67, 73–84], cryptographic engines and primitives [78–83, 85–98], and side-channel prevention [99–107] have been proposed. A systematic survey of the latest RISC-V security solutions will help the community understand the current state and future trends.

The organization of this article is as follows. First, we provide an overview of RISC-V security, focusing on discussing platform security requirements, the root of trust, and the building blocks of RISC-V architecture security, which lay a foundation for the enforcement of system and application security policies (**Section** 2). Then, as it is summarized in Figure 1, we categorize our survey into the following topics: hardware and physical security (**Section** 3.1), hardware-assisted security units (**Section** 3.2), memory protection (**Section** 4), ISA security extensions (**Section** 5), cryptographic primitives (**Section** 6), and protection against side-channel attacks (**Section** 7). Finally, we summarize our observations and discuss future directions of RISC-V security research (**Section** 8).

2 AN OVERVIEW OF RISC-V SECURITY

In this chapter, we outline the security requirements and foundations of embedded platforms. We discuss the application and system security requirements (Section 2.1), the hardware security module as the root of trust (Section 2.2), and the hardware and architectural foundations of the RISC-V security mechanism (Section 2.3).

2.1 Security Requirements of General Platforms

The goal of RISC-V is to become a general instruction set architecture. RISC-V has been used in low-power Internet of Things [108], storage controllers [4, 109], artificial intelligence machine learning [110–112], wireless sensor networks, data centers [113], high-performance computing [114], and many other application scenarios. Table 1 lists the representative RISC-V boards on the market, and summarizes the processor models, security features, and target application scenarios. We can see from the table that most of the RISC-V SoCs are still used in low-power IoT devices. Recently, SiFive cooperated with Intel to release the Performance P550 core, which can be scaled up to quad-core complicated configurations, using an area similar to that of a single Arm Cortex-A75, while providing significant performance advantages per area [115]. We can expect that more and more high-performance RISC-V chips will be available in the future. Many software and toolchains have been integrated into the RISC-V ecosystem. The RISC-V GNU Compiler Toolchain [116] and SiFive Freedom Studio IDE Toolchain [117] are two representative ones. Table 1 also lists operating systems that have supported the

Table 1. Representative RISC-V boards and their main features, application scenarios and operating system support.

Platforms	SoC & Processor	Privilege Mode	Security Feature	Arm Peer	Target App	Operating System Support
ICE EVB	XuanTie C910; 64-bit Dual cores 1.2GHz	U+S+M	PMP	Cortex-A55	5G, AI, Mobile	Linux, Android
HiFive Unmatched	SiFive U740; 64-bit Quad cores 1.4GHz	U+S+M	In-order; PMP	Cortex-A55	Generic PC	Linux
HiFive Unleashed	SiFive U54; 64-bit Quad cores 667MHz	U+S+M	In-order; PMP	Cortex-A53	AI, IoT	Linux, VxWorks
BeagleV	SiFive U74; 64-bit Dual cores 1.0GHz	U+S+M	PMP	Cortex-A55	AI	Linux, Zephyr
PolarFire Icicle	SiFive U54; 64-bit Quad cores 667MHz	U+S+M	In-order; PMP	Cortex-A53	AI, IoT	Linux, seL4
Kendryte KD233	Kendryte K210; 64-bit Dual cores 400MHz	M	AES and SHA Accelerator	Cortex-M7	AI, IoT	FreeRTOS
HiFive1 RevB	SiFive E310; 32-bit core 320MHz	M	In-order; PMP	Cortex-M4	IoT	Bare-metal, embOS, FreeRTOS, Mynewt, RT-Thread, Zephyr
Gigadevice RV-STAR	GD32VF103; 32-bit core 108MHz	M	NA	Cortex-M3	Low-Power	Bare-metal

RISC-V boards. Linux and FreeRTOS are two important ones. In addition, many RISC-V SoCs can run bare-metal applications, which do not depend on an operating system.

RISC-V and Arm architecture compete with each other for similar application scenarios. Mobile Internet is an important application scenario of Arm devices. Applications running on mobile devices such as smartphones and tablets increasingly rely on machine learning services to optimize user experience, such as estimating battery life based on user behavior, improving image quality, or performing voice recognition [25]. These services require frequent interaction with cloud servers, and the high sensitivity of such remotely processed data causes billions of users to face serious privacy risks. Recently, a British government contractor database containing more than 1 million fingerprints and facial recognition information was leaked [118], posing a major challenge to user privacy. Clients and service providers can use encryption technologies such as homomorphic encryption (HE) [119] and secure multi-party computing (SMPC) [120] to securely process private inputs under encryption, or use provable security protocols to jointly compute any function on private inputs. Unfortunately, in the networking scenario the computational and network communication bottlenecks of performing complicated machine learning tasks greatly limit the usefulness of the above technologies. Processing all sensitive user data on-premise not only reduces the risk of data leakage, but also improves the performance of data processing. Therefore, exploring hardware-assisted solutions to provide secure and private complicated computing services directly on mobile devices is an important application requirement.

Due to the risk of tampering with system executable files via physical attacks, trusted boot is critical for system life cycle security. One of the principles of building a secure system is to generate a chain of trust from all software parts between the first bootloader to the last trusted application [36]. This chain of trust is based on a root of trust (RoT) that will never be easily tampered with. This is called the secure boot sequence. Many security devices including laptops, desktops, smart phones, and IoT devices, need to implement secure boot to ensure system integrity. The secure boot architecture is complicated, relying on code verification units to ensure the integrity of the chain of trust. The public key cryptography such as elliptic curve digital signature algorithm (ECDSA) and secure hash algorithm (SHA) are the basic primitives of secure boot, which are usually implemented in the RoT such as a hardware security module (HSM). We will discuss HSM and RoT in Section 2.2.

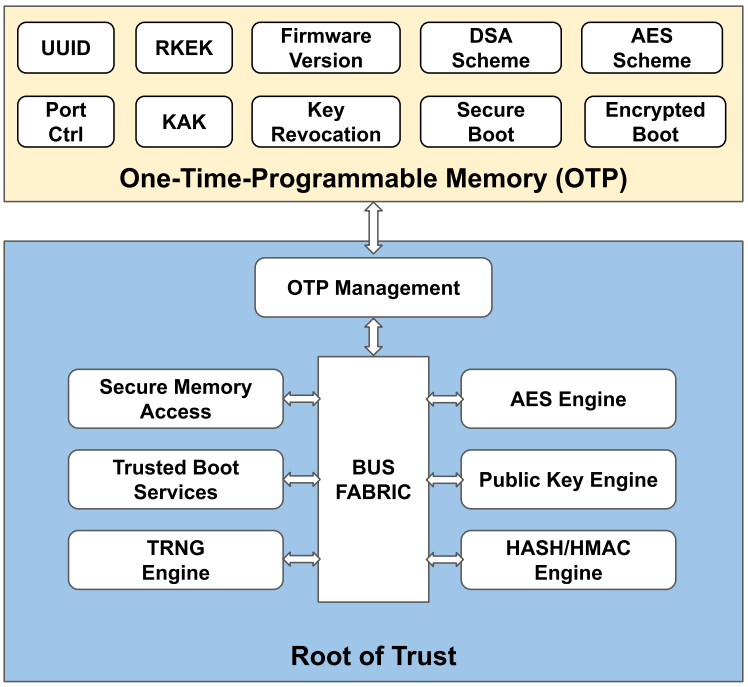


Fig. 2. The Building Blocks of the Root of Trust or Hardware Security Module.

Privilege management is the fundamental mechanism of architecture security. If the system can control tasks to run in privileged or non-privileged mode, and limit the task's access to resources such as RAM, executable code, and peripherals, it will make the microcontroller application more secure. For example, preventing certain code from executing in RAM can prevent attacks including buffer overflows and malicious code execution. However, implementing a memory protection mechanism [121] will make application design more complicated, because memory protection needs to determine the memory area limits and describe these limits to the operating system. Also, the memory protection mechanism requires differentiating operations and restrictions of applications. The memory protection strategy that restricts each task to its own memory area may be the safest, but the design and implementation are also the most complicated. Trusted Execution Environment (TEE) solutions for protecting sensitive code and data are widely deployed. Major CPU vendors have introduced their TEEs, such as Arm TrustZone [50] and Intel SGX [122] to enable platform security zones. There are many application scenarios for TEE, including cloud servers, mobile phones, ISPs, IoT devices, sensors, and hardware tokens. TEE needs to be implemented based on hardware security building blocks including privilege management, memory protection, and even trusted boot [41, 71]. We will outline RISC-V security building blocks in the rest of this Section.

2.2 The Root of Trust and the Hardware Security Module

The root of trust (RoT) [123] is the foundation on which all secure operations of computing systems depend. It contains keys for encryption functions, supports trusted boot and TEE. RoT is also important for public key infrastructure (PKI), which is used to generate and protect root and certificate authority keys, sign code for software security, immutability and authenticity, and create digital certificates for identity verification. Because the security of the system relies on the keys used to encrypt and decrypt data, as well as digital signatures and signature verification functions, the RoT is the always trusted source in the encryption system. RoT can secure data and applications and help build the chain of trust in the entire ecosystem.

The RoT must be secure by design. The hardware-based RoT will not be attacked by malicious software, so it is the most secure. The RoT can be an independent security module or a security module in a system-on-chip (SoC). A fixed-function RoT is a state machine designed to perform a specific set of operations, such as data

encryption, certificate verification, and key management. Usually these functions are static and can only perform their specially designed functions. In addition, there is a kind of programmable RoT. It is built around the CPU, can perform more complicated security functions, can be upgraded, and can run new encryption algorithms and security applications to counter evolving attack vectors.

Since the RoT is the target of attackers, it is usually executed in isolation to ensure that sensitive security functions are executed in a dedicated security domain physically separated from the general-purpose processor. Safely isolating security functions in a physically separated RoT can reduce architecture complexity and optimize CPU performance. The RoT shall also have comprehensive anti-tampering and side-channel resistance capabilities, prevent fault injection and side-channel attacks, and support layered security to provide multiple layers of strong defense. For hardware-based roots of trust [124], encryption engines, keys, and other sensitive security resources can only be accessed in hardware. Based on the hardware RoT, software security mechanisms can be implemented to provide additional flexibility.

RoT solutions usually include a hardened hardware security module (HSM) that generates and protects keys and performs encryption functions in its secure environment [125]. HSM is a tamper-proof hardware device that can enhance system security. The HSM is usually used for platforms with high data security and trust, which is inaccessible outside the system, so the system can trust the authentic and authorized keys and other encrypted information received from the HSM. The HSM can pass various FIPS certifications to prove its security specifications. The implementation of the HSM and the RoT is complicated, involving hardware and architecture, including system permission level control, secure memory access, password instructions, random number generators, etc. As shown in Figure 2, the RoT or HSM usually includes the following main components:

- **OTP Management Module** manages one-time programmable (OTP) memory [126], which is non-volatile and used to store keys and other security assets. OTP can be realized based on semiconductor anti-fuse and MOS gate oxide breakdown anti-fuse. OTP memory can only be programmed once, which is an irreversible process, thus ensuring security. The original equipment manufacturer (OEM) programs the OTP before the chip leaves the factory, and through it can write important trust-sensitive data, such as UUID, OEM key, firmware version, and trusted boot-related schemes, policies, and configuration parameters. OTP bears the RoT information in an immutable way to support the chain of trust throughout the chip life cycle.
- **Secure Memory** features multiple interfaces and a hardened memory protection unit. The RoT's RAM stores secure assets in a memory region isolated from the rest of the system. It may also include a small amount of ROM. Access to the memory regions is protected by the MPU[127] or PMP [128] mechanisms to guarantee only entities with proper privilege levels can access the protected memory areas. We will discuss RISC-V secure memory related research in Section 4.
- **Symmetric Cryptographic (eg. AES) Engine** is used for message and image file encryption and decryption to guarantee data confidentiality and support secure boot [129]. Symmetric encryption was the only type of encryption in use prior to the development of asymmetric cryptography in the 1970s. It remains by far more widely used because of higher performance and simpler key management. The Advanced Encryption Standard (AES) is a specification for the encryption of electronic data established by the U.S. National Institute of Standards and Technology (NIST) in 2001 [130]. Since then, AES has become the most widely used symmetric encryption algorithm. We will discuss RISC-V ISA extensions for AES in Section 5.1.
- **Asymmetric Cryptographic [131] (eg. Public Key RSA) Engine** is used for message encryption and enables the sender to combine a message with a private key to create a short digital signature on the message. This scheme has the advantage of not having to share symmetric keys while gaining the higher data throughput advantage of symmetric-key cryptography. Asymmetric cryptographic systems use key pairs: a public key that may be known to others, and a private key that only the owner knows. The generation of this key pair depends on a cryptographic algorithm based on a mathematical one-way function. Effective security requires that the private key be kept private. The public key can be distributed. In such a system, anyone can use the public key of the target recipient to encrypt the message, but can only use the private key of the recipient to decrypt the encrypted message. Public key encryption can also perform reliable authentication by creating a short digital signature on the message. Public key algorithms are the basic security primitives in modern

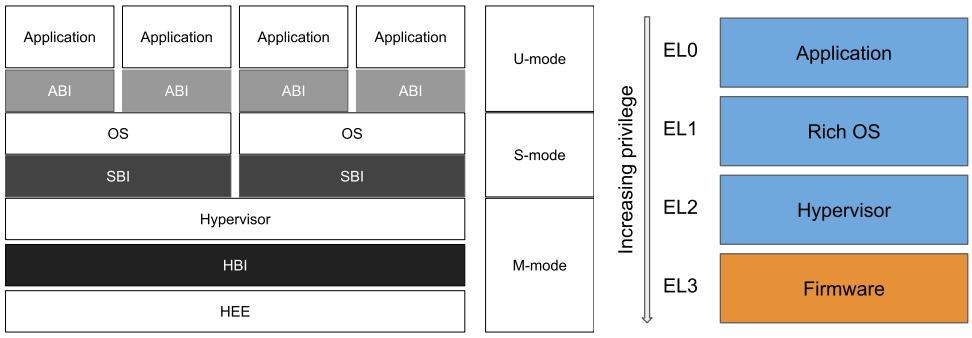
cryptographic systems, including applications and protocols that can guarantee the confidentiality, authenticity and non-repudiation of electronic communications and data storage. They are the basis of many Internet standards, such as the transport layer security protocol. Some public key algorithms provide key distribution and confidentiality (for example, Diffie-Hellman key exchange [132]), some provide digital signatures, and some provide both (for example, PKCS [133]). Compared to symmetric encryption, asymmetric encryption is much slower than good symmetric encryption, which is too slow for many purposes. Today's cryptographic systems such as TLS use both symmetric encryption and asymmetric encryption. Asymmetric Cryptography is an essential building block for secure boot.

- **HASH/HAMC Engine** is used for message hashing to guarantee data integrity and support secure boot. A hash function accepts a variable-length block of data as input and produces a fixed-size hash value, which can be used for message authentication, digital signatures, one-way password, and intrusion detection etc. Hash-based message authentication code (HMAC) [134] is a specific type of message authentication code, involving cryptographic hash functions and secret keys. Like any MAC, it can be used to simultaneously verify data integrity and message authenticity. HASH and HMAC play an important role in various security applications and Internet protocols. In addition, the hash function is a necessary part of key derivation and public key algorithms such as PKCS [133] and ECDSA [135].
- **Random Number/Bit Generation (eg. TRNG [136]) Engine** generates random numbers or bits for multiple cryptographic algorithms and protocols. Many network security algorithms and protocols based on cryptography use random values. For example, key distribution and mutual authentication schemes, session key generation, key generation for RSA public key encryption algorithm, and bit stream generation for symmetric stream encryption. There are two fundamentally different strategies for generating random bits. One strategy is to generate each bit based on an unpredictable physical process. This type of random bit generator (RBG) is usually called a non-deterministic random bit generator (NRBG). Another strategy is to use algorithms to calculate bits deterministically. This is called a deterministic random bit generator (DRBG) [137]. The DRBG algorithm generates a bit sequence according to an initial value determined by a seed, which is determined by a seed determined according to the output of the randomness source. The seed used to instantiate DRBG must contain enough entropy to ensure randomness. If the seed is kept secret and the algorithm is properly designed, the bits output by DRBG will be unpredictable. We will discuss RBG related instruction set architecture and algorithm research in Section 5.1 and Section 6, respectively.
- **Trusted Boot Services** reduce the risk of firmware rootkits. It starts with a first-stage Boot ROM that is synthesized into gates. A device with secure boot [129, 138] enabled will first verify whether the firmware is digitally signed when it starts, and the firmware will check the digital signature of the bootloader to verify that it has not been modified. The bootloader of a trusted boot-enabled device verifies its digital signature before loading the kernel. The kernel sequentially verifies all other components in the boot process, including boot drivers and boot files. If the file has been modified, the bootloader will detect the problem and refuse to load the damaged component. RoT enables trusted servers on the network to verify the integrity of the system boot process. The trusted boot services can be implemented in the RoT, but the service routines are mainly related to system runtime. In this article, we will not discuss trusted boot services in detail. We will discuss them in our next survey of the RISC-V system and application security, which will be a companion to this article.

2.3 Building Blocks of RISC-V Architecture Security

2.3.1 RISC-V Architecture Stacks and Privilege Modes. RISC-V can support different software stack implementations. A simple system can be a bare-metal application running on an application execution environment (AEE) in the machine mode (M-mode). The application runtime interacts with a particular application binary interface (ABI), which includes the supported user-level ISA and a set of ABI calls to interact with the AEE. The ABI hides details of the AEE from the application, providing an abstract layer for flexibility of implementing the AEE.

RISC-V can also run an operating system (OS) that can support multiple applications. Each application communicates over an ABI with the OS, which provides the AEE. RISC-V operating systems interface with a supervisor execution environment (SEE) via a supervisor binary interface (SBI). An SBI comprises the user-level



(a) RISC-V. (b) Armv8-A.

Fig. 3. Privileged software stack and corresponding privileged execution modes.

and supervisor-level ISA together with a set of SBI function calls. Using a single SBI across all SEE implementations allows a single OS binary image to run on any SEE. The SEE can be a simple bootloader and BIOS-style IO system on a low-end hardware platform, or a virtual machine in a high-end server, or a thin translation layer over a host operating system in an architecture simulation environment [52].

As Figure 3 shows, RISC-V can run a virtual machine monitor configuration where multiple OSs are supported by a hypervisor. This is a typical deployment in complicated infrastructure as a service scenarios. Each OS communicates via an SBI with the hypervisor, which provides the SEE. The hypervisor communicates with the hypervisor execution environment (HEE) using a hypervisor binary interface (HBI) to isolate the hypervisor from the hardware platform.

At any time, a RISC-V hardware thread (hart) is running at some privilege level encoded as a mode. Three RISC-V privilege levels are currently defined: **User(U) mode** (level 0), **Supervisor(S) mode** (level 1), and **Machine(M) mode** (level 3) [52]. There was a *level 2 H-mode* defined in the RISC-V privileged architecture. H-mode was removed in version 1.10 to enable recursive virtualization support in S-mode. For backward compatibility, the latest RISC-V specification reserves the H-mode. In summary:

- The privilege level is used to provide differentiated protection for different components of the software stack, which lays the foundation for the security of the RISC-V platform.
- M-mode is the highest privilege level. It is the only mandatory privilege level of the RISC-V hardware platform. Code running in M mode is usually inherently trustworthy because it has low-level access to the machine implementation. M-mode can be used to manage the secure execution environment on RISC-V.
- Many RISC-V implementations support U-mode to protect the rest of the system from application code.
- S-mode can be added to provide isolation between a supervisor-level operating system and the SEE.

A hart normally runs application code in U-mode until some trap such as a supervisor call or a timer interrupt forces a switch to a trap handler, which usually runs in a more privileged mode. The hart will then execute the trap handler, which will eventually resume execution at or after the original trapped instruction in U-mode. Traps that increase privilege level are termed vertical traps, while traps that remain at the same privilege level are termed horizontal traps. The RISC-V privileged architecture provides flexible routing of traps to different privilege levels. Each privilege level has a core set of privileged ISA extensions with optional extensions and variants. The M-mode supports an optional standard extension for physical memory protection (PMP) [128], which is an important security enabler of RISC-V.

The RISC-V privilege level is a similar concept as the ARM exception level. As Figure 3 shows, the Armv8-A architecture allows implementations to choose whether to implement all exception levels, and select the allowed execution state for each implemented exception level[139]. EL0 and EL1 are the exception levels that must be achieved. EL2 and EL3 are optional. The choice not to implement EL3 or EL2 is of great significance. EL3 is the only level that can change the security status. If the implementation chooses not to implement EL3, the PE will not be able to access a single security state. EL2 contains many virtualization functions. No implementation of EL2 can omit these functions. All current Armv8-A implementations support all exception levels [139], because

Table 2. Comparison of RISC-V PMP and ARM MPU Main Features.

	RISC-V PMP	**ARM MPU**
The smallest region size	4 Bytes	32 Bytes
The maximum size of a region	32 GB (if XLEN = 32)	4 GB
Region granularity	Configurable (2^{G+2} Bytes, $G \geq 0$)	32 Bytes
Privileged and unprivileged settings	Hybrid (If PMP configuration register L bit is set, the setting also applies to M-mode)	Independent (Explicitly indicated by the MPU_RBAR AP field)
Supported memory attributes	R/W/X	R/W/X
Maximum number of supported memory regions	16 (All for unprivileged, some also applies to privileged if L bit is set)	16 (8 for privileged, 8 for unprivileged)

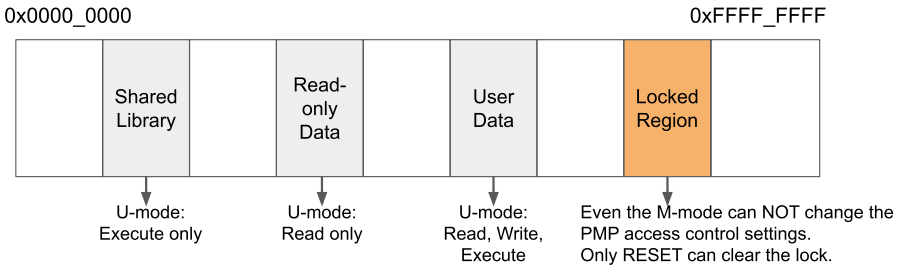


Fig. 4. Demonstration of RISC-V physical memory protection.

most standard software requires these exception levels. The implementation can also choose the execution state that is valid for each exception level. If AArch32 is allowed at the exception level, it must be allowed at all lower exception levels. For example, if EL3 allows AArch32, it must be allowed at all lower Exception levels. However, existing implementations also have limitations. For example, Cortex-A32 only supports AArch32 for all exception levels. Some modern implementations, such as Cortex-A55, implement all exception levels, but only allow EL0 to use AArch32, exception level EL1, EL2 and EL3 must use AArch64.

2.3.2 Physical Memory Protection (PMP). For security control, an optional PMP [52, 128] unit provides per-hart machine-mode control registers to allow physical memory access privileges (read, write, execute) to be specified for each physical memory region. The PMP values are checked in parallel with the physical memory attribute checks. In effect, PMP can grant permissions to S and U modes and can revoke permissions from M-mode, which by default has full permissions. PMP violations are always trapped precisely at the processor. From the viewpoint of functions, RISC-V PMP is the equivalent of ARM MPU [127], which is a programmable unit that allows privileged software to define memory access permissions for separate memory regions. RISC-V PMP and ARM MPU are very similar, but some of their critical configurations such as region size and number of supported regions are different. We compare the key feature set of RISC-V PMP and ARM MPU in Table 2. We summarize the main characteristics of RISC-V PMP as follows:

- PMP checks are applied to all accesses when the hart is running in **S** or **U** modes.
- PMP checks are applied to loads and stores when the *MPRV* bit is set in the *mstatus* register and the *MPP* field in the *mstatus* register contains **S** or **U**.
- PMP checks are applied to page-table accesses for virtual-address translation, for which the effective privilege mode is **S**.
- PMP checks may additionally apply to **M** mode accesses, in which case the PMP registers themselves are locked, so that even **M** mode software cannot change them without a system reset.
- The standard PMP encoding supports regions as small as four bytes.
- RISC-V can maximally support setting sixteen PMP regions.

PMP is mainly used to prevent hart running in lower privilege levels such as U and S modes from accessing privileged memory contents. For example, a regular user hart should be prevented from modifying or even reading the data of shared libraries. Therefore, the memory region in which the shared library data reside can be set as *execute only* using PMP. Even if a hart is running in M-mode, in some cases it is beneficial to prevent it from modifying platform related configurations that may cause runtime errors. By setting the *L* bit of the PMP configuration register, PMP can apply the memory settings to M-mode to enforce protection policies and lock the memory region to prevent the M-mode hart from changing the enforced PMP settings. Figure 4 demonstrates the effect of PMP for runtime memory protection.

We verified RISC-V PMP settings in M-mode on a *HiFive1 Rev B* board. The reformatted source code is demonstrated as Listing 1. The *line#38* tries to write data into a PMP write-disabled target memory address, which triggers *store/AMO access fault* (exception code 7).

Listing 1. Verification of the PMP feature on HiFive1 Rev B board [3]. The following codes demonstrate how to configure a PMP Region. The code is adopted from example-pmp.

```
1  /* This source code is released under Apache2 and MIT licenses.*/
2  // Declare Global Variables
3  int main()
4  {
5          // Declare Local Variables
6          /* The "protected_global" is the target PMP memory address
7           * Set the address to be 4-byte aligned and region size to be NAPOT_SIZE bytes */
8          size_t protected_addr = ((size_t) &protected_global) >> 2;
9          protected_addr &= ~(NAPOT_SIZE >> 3);
10         protected_addr |= ((NAPOT_SIZE >> 3) - 1);
11
12         /* Initialize interrupt handling on the current hardware thread */
13         cpu = metal_cpu_get(metal_cpu_get_current_hartid());
14         cpu_intr = metal_cpu_interrupt_controller(cpu);
15         metal_interrupt_init(cpu_intr);
16
17         /* Register the function store_access_fault_handler as the
18          * handler to process cpu exceptions. */
19         rc = metal_cpu_exception_register(cpu, ECODE_STORE_FAULT, store_access_fault_handler);
20
21         /* Reset/initialize the PMP unit */
22         pmp = metal_pmp_get_device();
23         metal_pmp_init(pmp);
24
25         /* Disable write and execution to protected_global.
26          * The PMP region is locked so it takes effect on M-mode. */
27         struct metal_pmp_config config = {
28                 .L = METAL_PMP_LOCKED,
29                 /* Set the region's upper bound to be naturally-aligned power of two,
30                  * which is determined by the value of A.*/
31                 .A = METAL_PMP_NAPOT,
32                 .X = 0, .W = 0, .R = 1,
33         };
34         rc = metal_pmp_set_region(pmp, 0, config, protected_addr);
35
36         /* Attempt to write to protected_global. This should trigger a store
37          * access fault exception and enter the registered handler. */
38         protected_global[0] = 6;
39
40         // Execution shall not arrive at this point if PMP setting is successful
41         return 0;
42 }
```

Listing 2. Demonstration of *AESE* and *AESD* instructions defined in Armv8 Cryptographic Extension [140]. Currently, the standardisation of RISC-V AES and other cryptographic instructions is on-going.

```
/* AES single round encryption */
AESE <Vd>.16B, <Vn>.16B
{
    bits(128) operand1 = V[d];
    bits(128) operand2 = V[n];
    bits(128) result;
    result = operand1 EOR operand2;
    result = AESSubBytes(AESShiftRows(result));

    V[d] = result;
}

/* AES single round decryption */
AESD <Vd>.16B, <Vn>.16B
{
    bits(128) operand1 = V[d];
    bits(128) operand2 = V[n];
    bits(128) result;
    result = operand1 EOR operand2;
    result = AESInvSubBytes(AESInvShiftRows(result));

    V[d] = result;
}
```

2.3.3 Cryptographic Instruction Set. Securely and efficiently performing cryptographic operations is a basic requirement for a wide range of computing platforms. The dedicated instruction set extension (ISE) is often used to achieve this purpose. The implementation of the cryptographic instruction set has two advantages over the software-based implementation. First, the implementation based on CPU hardware can maximize the performance of cryptographic operations. Second, hardware implementation can hide implementation details and reduce the attack surface. The Armv8 Cryptographic Extension provides instructions for the acceleration of AES, SHA, Polynomial Multiply, SM3, and SM4 [140]. Listing 2 demonstrates the *AESE* (AES Encryption) and *AESD* (AES Decryption) instructions defined in Armv8 Cryptographic Extension. A single execution of these instructions can conduct a round of AES encryption or decryption operation, which actually consists of multiple execution steps.

RISC-V standard cryptographic ISE definition is still ongoing work. There are some active research projects on this topic. Stoffelen et al. [98] present the first optimized assembly implementations of table-based AES, bitsliced AES, ChaCha, and the Keccak-f[1600] permutation for the RV32I instruction set. Marshall et al. [82] further recommend separate ISEs for 32 and 64-bit RISC-V base architectures, with measured performance improvements for an AES-128 block encryption of 4× and 10× with a hardware cost of 1.1K and 8.2K gates when compared to a software-only implementation using T-tables. Some RISC-V SoCs such as the PolarFire SoC FPGA have adopted heterogeneous architecture to integrate Arm-based co-processors as the cryptographic engine. We will further discuss the RISC-V cryptographic ISE research in Section 5.1.

2.3.4 Instruction Pipeline. Instruction pipeline technology divides instructions into a series of sequential steps executed by different processor components, and processes different instructions in parallel in a single processor [141], so that all processor components are fully utilized. Out-of-order execution is a paradigm used in most high-performance processors to take advantage of instruction cycles that would otherwise be wasted. In this paradigm, the processor executes instructions based on the availability of input data and execution units rather than the original order in the program. Thus, the processor can avoid being idle while waiting for the completion of the previous instruction, and at the same time can immediately process the next instruction that can independently run. As a result, out-of-order execution [142] is an indispensable performance feature of many modern processors. When an out-of-order execution reaches a conditional branch instruction, its direction depends on the instructions that have not yet completed execution. In this case, the processor can checkpoint its current register state, predict

the path that the program will follow, and speculatively execute instructions along that path. If the prediction is correct, the checkpoints are not useful, and the instructions are cancelled in the order of program execution. Otherwise, when the processor determines that it follows the wrong path, it will discard all pending instructions along the path by reloading its state from the checkpoint, and resume execution along the correct path to ensure the correctness of the program logic state. The branch predictor tries to guess which way the branch will go before the certainty is known. The purpose of the branch predictor is to improve the flow in the instruction pipeline. In many modern pipelined microprocessor architectures, branch predictors play a vital role in achieving high performance. However, out-of-order execution and speculative branch prediction lead to well-known Meltdown [20] and Spectre [21] vulnerabilities.

RISC-V ISA avoids over-defining a particular microarchitecture style (e.g., microcoded, in-order, decoupled, out-of-order) or implementation technology (e.g., full-custom, ASIC, FPGA). RISC-V allows efficient implementation in any of these styles [52]. Rocket Chip [143] is an open source Sysem-on-Chip design generator that emits synthesizable RTL. It uses the Chisel hardware construction language to form a complicated generator library for the core, cache, and interconnection to the integrated SoC. Rocket Chip generates general-purpose processor cores, and provides an in-order core generator (Rocket) and an out-of-order core generator (BOOM). Rocket Chip supports the integration of custom accelerators in the form of instruction set extensions, coprocessors or completely independent new cores. Rocket Chip has been taped out and produced a prototype capable of booting Linux. Gonzalez et al. [100] replicated Spectre attacks on a BOOM core. To mitigate the attack, they implement a small *L0 speculation buffer* that holds refill data from speculating load misses, and flushes the data when the load is resolved as misspeculated. This prevents misspeculated loads from affecting the state of the cache, while still allowing correctly speculated loads to broadcast their data into the rest of the machine as soon as possible to maintain performance. We will further discuss side-channel prevention related topics in Section 7.

In the rest of this article, we will discuss RISC-V hardware and architecture security by topic. Categorizing research papers is challenging, because some research involves multiple topics. For example, when we discuss instruction set extensions, there are related studies on side-channel prevention, but preventing side-channel attack itself is a major research topic, which poses a challenge to the categorization of research papers. In this case, we will discuss a specific study under the topic closest to its work.

3 HARDWARE SECURITY

Embedded devices such as IoT devices face the challenge of physical attacks through side channels or fault injection. Learning from the vulnerabilities of speculative execution, the design of computing architecture should consider security, not just performance. Although the semiconductor industry uses a variety of verification techniques to ensure system-on-chip (SoC) security, attacks are becoming more and more sophisticated. A series of actual attacks that have affected major hardware manufacturers in recent years have proved that ensuring chip security is extremely challenging. It is still a technological trend to reduce software TCB by increasing hardware security extensions. Security solutions such as encryption primitives or TEE based on the underlying TCB will continue to face many challenges in the future. In addition, the security architecture needs to strike an optimal trade-off between the application's high performance and low power consumption [63]. As a new architecture, the RISC-V community is implementing various security solutions. RISC-V's openness and instruction set extension capabilities provide unprecedented opportunities for innovation in the realization of chip security solutions. In this section, we summarize the state-of-the-art RISC-V security research in the hardware and architecture layer. We categorize the existing hardware security research into two topics: hardware and physical security and hardware-assisted security units.

3.1 Hardware and Physical Security

Hardware vulnerabilities [153] may be caused by unintentional design errors and maliciously implanted hardware Trojans during design or manufacturing. Incorrect design specifications, defective design implementation, or incorrect translation of design in RTL synthesis may all lead to design errors. Recent studies have confirmed that hardware vulnerabilities can be exploited by various attacks, including physical access attacks [147, 148],

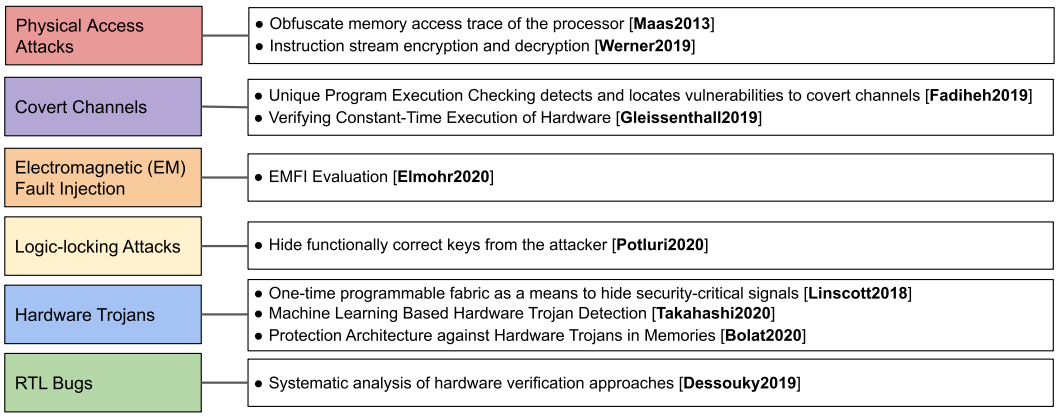


Fig. 5. RISC-V Hardware and Physical Security. *References:* ***Fadiheh2019****[144]*, ***Gleissenthall2019****[64]*, ***Elmohr2020****[145]*, ***Potluri2020****[146]*, ***Maas2013****[147]*, ***Werner2019****[148]*, ***Linscott2018****[149]*, ***Takahashi2020****[150]*, ***Bolat2020****[151]*, ***Dessouky2019****[152]*.

covert channels [20, 21], fault injection [145], and logical lock attacks [152]. In addition to leaking critical system or user information, a major risk of hardware vulnerabilities is its irreparability. After tapeout, firmware and software updates will not change the hardware runtime behavior of the chip, which may lead to product recalls. A major challenge in preventing hardware vulnerabilities is that current industry detection methods cannot achieve a good hardware vulnerability detection ratio. Therefore, exploring new hardware security and vulnerability detection technologies is a top priority. Figure 5 summarizes the latest RISC-V hardware and physical security research and shows a general view of hardware-related vulnerabilities and attack models.

3.1.1 RTL Bugs. Dessouky et al. [152] organized an international hackathon to show that the current hardware security verification technology is fundamentally limited and cannot detect RTL bugs that are common in real platforms and can lead to hardware vulnerabilities. Specifically, they injected 31 RTL bugs into two open source RISC-V SoC designs to synthesize different types of common hardware vulnerabilities. These bugs include incorrect privilege escalation, address overlapping, improper write permissions to certain system registers, insecure encryption functions, insecure key storage, and hard-coded passwords. More than 50 teams from all over the world have used formal verification, assertion-based simulation, software-based testing and even manual inspection methods to detect these bugs for months. The industry-leading formal verification technology only detected 15 bugs. The detection ratio of bugs that can lead to secret leakage is very low, which shows that further exploration to advance the technological progress of hardware security verification is extremely important. Sadeghi et al. [154] further summarize the lessons learned in these hardware security competitions, they observed that the main techniques for detecting hardware vulnerabilities include simulation-driven methodologies, information flow analysis, software-driven simulations, and checking RTL codes using Lint tools. They also envision that fuzzing hardware interfaces will be potentially an effective way to detect hardware bugs.

3.1.2 Hardware Trojans. The hardware Trojan (HT) is another major security risk that causes hardware vulnerabilities [155]. Theoretically, malicious engineers and chip manufacturers can modify the hardware design and implementation to include hardware backdoors so that the attacker can fully control the system. Hardware Trojan detection is an important defense mechanism. The development of defense mechanisms against these dangerous Trojans is relatively lagging behind. Hepp et al. [156] designed and integrated four hardware Trojans into a post-quantum encryption-enhanced RISC-V microcontroller. The microcontroller was taped out in September 2020. The impact of these HTs is multifaceted, from simple denial of service to side channel vulnerabilities, and the transmission of sensitive information to external observers. For each HT, they use design tools or simulations to estimate the detectability of these Trojans. Their preliminary observation is that some HTs are easily detected by design tools. However, some HTs that modify the software control flow, causing little interference, are not easy to detect. However, the use of these Trojans that modify the software control flow requires covert modifications

to the executable code, which increases the difficulty of using these Trojans to implement attacks in reality. This work provides realistic test equipment for hardware Trojan detection tools.

Linscott et al. [149] propose a novel architecture that maps the security-critical part of the processor design to a one-time programmable LUT-free structure. By analyzing the HDL of the target module, a programmable structure can be automatically generated. By letting the trusted party randomly select a mapping configuration for each chip, the proposed scheme can prevent an attacker from knowing the physical location of the target signal. In addition, they provide a decoy option to map security-critical signals to detect hardware Trojans that hit the decoy. Using this defense method, any Trojan that can analyze the entire configurable structure must use complicated logic functions and take up a large silicon area, which greatly increases the possibility of being detected by security tools. They evaluated the solution on the RISC-V BOOM processor and proved that by providing the ability to map each key signal to 6 different locations on the chip, the proposed scheme can reduce the attack success rate by 99% with an overhead of only increasing the area by 27%.

Side-channel detection [157, 158] is an effective method to discover potential hardware Trojans. It can measure any difference in system power consumption, electromagnetic (EM) emanation, and delayed propagation caused by Trojan insertion or modification in the real design to discover potential threats. However, these methods were evaluated on simple design prototypes such as the AES coprocessor. Moreover, the analysis methods used for these methods are limited by some statistical indicators, such as direct comparison of EM traces or T-test coefficients. Takahashi et al. [150] propose two new detection methods based on machine learning. The first method is to apply a supervised machine learning algorithm on the original EM trajectory to classify and detect hardware Trojans. Its detection rate is close to 90%, and the false negative rate is less than 5%. The second method is based on the outlier/novelty algorithm. This method is combined with the signal processing technology based on T-test, and has better performance. The detection rate is close to 100%, and the false positive rate is less than 1%. Takahashi et al. have evaluated the methods on the RISC-V general purpose processor. The area size ratios of the three hardware Trojans in the RISC-V processor are 0.53%, 0.27% and 0.1%, respectively. Although the inserted Trojans are small, the new methods can detect them.

Existing hardware Trojan research mainly focuses on Trojan attacks in logic circuits. There are still few studies on Trojan attacks in embedded memory. Hoque et al. [159] discuss a new hardware Trojan for embedded SRAM arrays. They demonstrate various types of Trojan circuits in SRAM including resistive short, bridge, and open in circuit nodes. These Trojans can evade industry-standard post-silicon memory testing and enable target data tampering after deployment. They can cause various malicious influences and have multiple activation conditions, have low overhead in power consumption, performance and stability, and incur negligible silicon area overhead. Bolata et al. [151] proposed a RISC-V microprocessor protection architecture against hardware Trojans in memory. The architecture is designed to detect the intrusion of hardware Trojans on the system instruction and data memory. The goal is to detect hardware Trojans that can force the microprocessor to run malicious code or read/write data in unauthorized memory locations. The proposed protection architecture relies on two checkers based on Bloom Filter that monitor the instructions fetched from the instruction memory and the access addresses in the instruction and data memory. They apply the protection architecture to a RISC-V FPGA microprocessor to run a set of software benchmarks for case study.

3.1.3 Logic-locking Attacks. Logic locking aims to solve the threat of IP piracy in the semiconductor supply chain. This technology adds a key gate with an input driven by a secret key to hide the internal details of the IP. Only when the programmed key is applied, the conversion is reversed to achieve the original function of the IP. Unfortunately, the existing logic lock function is constantly under attack, and it is difficult to achieve the desired goal. Although current attacks are mainly aimed at combinational circuits, these attacks can be extended to actual sequential circuits through scan-chains. Assuming the scan-input and scan-output ports are controllable and observable by an attacker. The attacker can selectively inject inputs to the scan-input port and analyze the responses from the scan-output port, thus functionally pirating the function of the protected IP module. A secure scan-chain is needed to prevent such attacks. Potluri et al. [146] observed that the flip-flop locking on the scan-input port can obfuscate functional output of the scan-output port. Thus, they propose SeqL, which isolates the functional path from the locked scan path and locks flip-flop inputs to achieve functional output corruption.

As a result, SeqL can hide the majority of the scan-correct keys which are functionally correct, thus maximizing the probability that the decrypted key observed by the attackers are functionally incorrect. They validated the effectiveness of the proposed solution on a fully-fledged RISC-V CPU and verified that SeqL can resist a broad range of attacks including SAT, Double-DIP, HackTest, SMT, FALL, Shift-and-Leak, and Multi-cycle attacks.

3.1.4 Electromagnetic Fault Injection. Electromagnetic fault injection (EMFI) technology is an important security challenge faced by embedded devices. Elmore et al. [145] proved through experiments that EMFI enables 320MHz RISC-V processors to skip or mistakenly handle instructions, thus confirming the possibility of attackers using EMFI to conduct widespread attacks. In addition, experimental results on Arm and RISC-V embedded processors show that EMFI attacks are more likely to succeed under lower power supply voltages and higher clock frequencies. They also observed that the exception code is useful for understanding the details of the injected fault, which provided further evidence that the instruction had been corrupted in many cases. Currently, there are still not many countermeasures against EMFI attacks.

3.1.5 Covert channels. Covert channels are another risk of sensitive information leakage. Meltdown and Spectre covert-channel vulnerabilities [20, 21] have been discovered in advanced processors, which have caused the public to be highly alert to hardware security. These covert channels can leak secret data without any explicit information flow between the secret and the attacker. It is generally believed that these covert channels are inherent in the advanced processor architecture based on speculative and out-of-order execution, and low-end processors do not have such security risks. However, Fadiheh et al. [144] show that covert channel information leakage is widespread, and they may also appear in average complexity processors with sequential pipelines. They propose a formal method called unique program execution checking (UPEC), which can systematically detect and locate covert-channel vulnerabilities. UPEC employs a formal analysis on the microarchitectural level RTL. UPEC defines the set of all state variables (registers, buffers, flip-flops) belonging to the logic part of the computing system's microarchitecture as the *microarchitectural state variables* of an SoC. It defines the subset of microarchitectural state variables that define the state of program execution at the ISA level excluding the program state that is represented in the program's memory as the *architectural state variables*. It defines the content of memory at a protected location as *secret data*. The UPEC property fails if the system under verification exists a state *soc state*, such that the transition to the next state *soc state'* depends on any *secret data*. Here, *soc state* and *soc state'* are vectors of state variables which include only architectural state variables. If any UPEC property failure is detected, then the design may contain covert channels. The effectiveness of UPEC was explored by targeting different design variants of the open-source RISC-V SoC generator Rocketchip [160].

Timing channels [161] can also leak sensitive information. Due to complicated fast paths and optimization functions, timing channels are difficult to avoid in modern hardware. A promising way to avoid timing channels is to design and verify conditions under which hardware designs can be executed in constant time. Gleissenthall et al. [64] propose *IODINE*, an accurate clock and constant time method that can eliminate timing channels in hardware. IODINE [64] defines a syntax to translate Verilog code to intermediate code using its language interpreter *VINTER*. Then it executes the intermediate language for timing channel analysis. For a predefined input vector, the analysis is cycle by cycle. In each cycle, it checks the influence set of each variable. For different input vectors, it expects that the influence set of any variable in these vectors in each loop is the same, otherwise there may be cases where the execution time of different test vectors is not constant. In view of the hardware circuit described in Verilog, including a set of sources and sinks and a set of specification assumptions, IODINE allows developers to automatically synthesize proofs to ensure that the hardware executes in a constant time. In other words, under a given usage assumption, the time taken from the source to the sink has nothing to do with the operands, processor flags, and interference from concurrent calculations. By using IODINE, encryption hardware designers can ensure that the hardware execution time is not secret value dependent, so their encryption engine will not leak secret keys. Similarly, CPU designers can ensure that programs (such as cryptographic algorithms) are executed in constant time with the correct structure. Two real bugs were detected by this methodology, one was in the FPU and the other was in the RSA encryption module. A main contribution of this research is that it proposes a formal methodology to verify the existence of timing side channels in hardware design in a deterministic way.

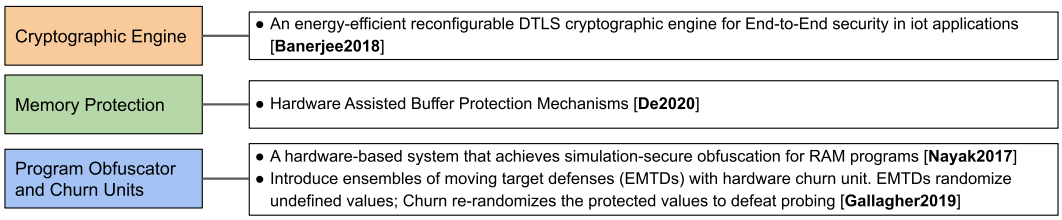


Fig. 6. Hardware-assisted Security Units. *References:* ***Nayak2017****[85],* ***Banerjee2018****[60],* ***Gallagher2019****[163],* ***De2020****[65].*

3.1.6 Physical Access Attacks. Attackers with physical access to a chip may directly probe the pin signals to observe sensitive information [162]. In order to prevent such attacks, the prior art proposes a secure processor to automatically encrypt and check the integrity of all data outside the processor, including data in DRAM and non-volatile memory. Although the security processor encrypts the memory content, DRAM transfers memory addresses in clear text on the memory bus. Attackers can snoop on the memory bus and observe the physical memory addresses accessed, and then collect sensitive data, such as encryption keys or information about user-level programs. To prevent such information leakage, it is necessary to make the memory address traces indistinguishable. Maas et al. [147] propose a new hardware architecture for effective oblivious computation that can ensure data confidentiality and memory trace obliviousness. In other words, since each memory access of the application program will cause the physical memory address of the accessed data to randomly change, the data access will not produce a fixed address pattern on the physical memory. Therefore, the attacker cannot probe any information about the DRAM locations accessed. Also in order to prevent physical access attacks, Werner et al. [148] proposed a method to protect RISC-V processors from fault injection attacks. They demonstrated the protection of control flow through instruction stream encryption and decryption, and the protection of conditional branches by adding redundancy to comparison operations and entangling the comparison result with the encrypted instruction stream.

3.2 Hardware-assisted Security Units

Hardware-assisted security units try to ensure a level of security that software cannot provide. The basic assumption of hardware-assisted security mechanisms is that hardware is less likely to have exploitable vulnerabilities than software. The hardware function can reduce the complexity of the software part to improve platform performance. Therefore, the industry is advocating a full range of hardware-assisted security methods including trusted computing, random number generation, crypto acceleration, malware detection [164]. At the same time, the academic community has also proposed many security solutions based on these industrial hardware trust anchors. The hardware-assisted security unit can be implemented in different forms in many scenarios. For example, in order to achieve the end-to-end security of IoT applications, the hardware unit can be implemented as an encryption engine. In the case of an ensemble of moving target defense, it can be realized as a churn unit. Hardware-assisted units can also be implemented for memory protection and transaction approval. RISC-V TEE including *TIMBER-V* [71] and *Keystone* [41] are created based on the memory protection mechanism and trusted boot service. *TIMBER-V* is based on the memory protection unit and tagged memory mechanism, which we will discuss in Section 4.1. *Keystone* relies on a trusted boot service, which is similar to Arm TEE, and is mainly related to system runtime, which will not be discussed in this article. We plan to discuss TEE as a major topic in our next RISC-V security survey article on systems and applications. Figure 6 summarizes the research of Hardware-assisted security units that we will discuss in the rest of this subsection.

3.2.1 Program Obfuscator and Churn Units. It is challenging to implement virtual black box (VBB) obfuscation of general programs in a pure software manner. Nayak et al. [85] proposed *HOP*, which uses secure hardware to realize the simulated security obfuscation of RAM programs. HOP only trusts the hardware processor. The theoretical analysis of HOP considers all the optimizations used in the actual design, including the use of hardware Oblivious RAM (ORAM), hardware scratchpad, instruction scheduling technology, and context switching. They introduced the FPGA prototype hardware implementation of HOP. Through various benchmark evaluations, the

cost of HOP is 8 to 76 times that of an insecure system. Compared with all previous efforts to achieve obfuscation (unimplemented), HOP has improved performance by more than three orders of magnitude, making obfuscation technology one major step closer to achieving the goal of being deployable in practice.

Constantly obfuscating the information required by the attacker is an effective counter-attack method. Frequent obfuscation will produce high system overhead. Gallagher et al. [163] proposed *Morpheus*, which is an ensemble of mobile target defense with a hardware churn unit, in which each mobile target defense uses hardware support to provide more randomness at a lower cost. When used in conjunction with obfuscation, Morpheus defense can provide powerful protection against control flow attacks. Security testing and performance research show that Morpheus has achieved high coverage protection against various control flow attacks, including protection against advanced attacks. In addition, a churning period of up to 50 milliseconds is at least 5000 times faster than the time required to penetrate Morpheus.

3.2.2 Memory Protection. Code injection and code reuse attacks such as buffer overflow and return-oriented programming (ROP) are still threats to RISC-V programs. De et al. [65] proposed two hardware security extensions for RISC-V. First, they use a physical unclonable function (PUF)-based random canary generation technology, which eliminates the need to store sensitive canary words in memory or CPU registers, so it is more secure and efficient. They implemented the proposed Canary engine in RISC-V Rocket Chip. The simulation results show that for a single buffer protection, the average execution overhead is 2.2%. When the protection is extended to all buffers, increasing the buffer count by 10 times will only increase the overhead by 1.5 times. Second, the author implements Fixer, a dedicated security coprocessor extension for flow integrity. FIXER enforces fine-grained control flow integrity (CFI) for programs running on the backward edge (return) and forward edge (call) without requiring any architectural changes to the processor core. Compared with software-based solutions, FIXER reduces energy consumption by 60% with minimal execution time (1.5%) and area (2.9%) overhead.

3.2.3 Cryptographic Engines. Datagram Transport Layer Security (DTLS) is an important protocol for end-to-end IoT communication security. The high computational overhead makes pure software DTLS implementation too costly for resource-constrained embedded devices. Banerjee et al. [28, 60] demonstrate the first hardware implementation of the DTLS protocol. The key component of the design is the reconfigurable element field elliptic curve encryption (ECC) accelerator, which is 238 times and 9 times more energy efficient than software and the latest hardware implementations. The complete hardware implementation of the DTLS 1.3 protocol is 438 times more energy-efficient than software, and the code size and data memory footprint are as low as 8KB and 3KB, respectively. Benchmarking of applications other than DTLS shows that the combination of a cryptographic accelerator and an on-chip low-power RISC-V processor can save up to two orders of magnitude of energy. Their test chip is made of 65nm CMOS. At 16MHz and 0.8V, each handshake consumes 44.08 μJ and each byte of encrypted data consumes 0.89 nJ.

Co-design of software and hardware can significantly improve the performance of cryptographic algorithms. Wang et al. [165] proposed the software and hardware co-design of the hash-based post-quantum signature scheme XMSS on the RISC-V embedded processor. They provide software optimizations for the SHA-256 parameter set and the XMSS reference implementation of multiple hardware accelerators, allowing area usage and performance to be balanced according to individual needs. Compared with pure software implementation, by integrating the hardware accelerator into the RISC-V processor, the key pair generation performance can be improved by more than 54 times. The signature generation time is less than 10 milliseconds, and the verification time is less than 6 milliseconds, which is 42 times and 17 times faster than software. They tested and measured the number of cycles on Intel Cyclone V SoC FPGA. The integration test of their XMSS accelerator and embedded RISC-V processor shows that the hash-based post-quantum signature can be actually used in a variety of embedded applications.

4 MEMORY PROTECTION

Retrospecting the history of computer engineering, many security vulnerabilities originated from two aspects. First, mainstream processor architectures and C/C++ language abstractions have only provided coarse-grained virtual memory-based protection since the 1970s. Second, the mainstream engineering methodology follows the

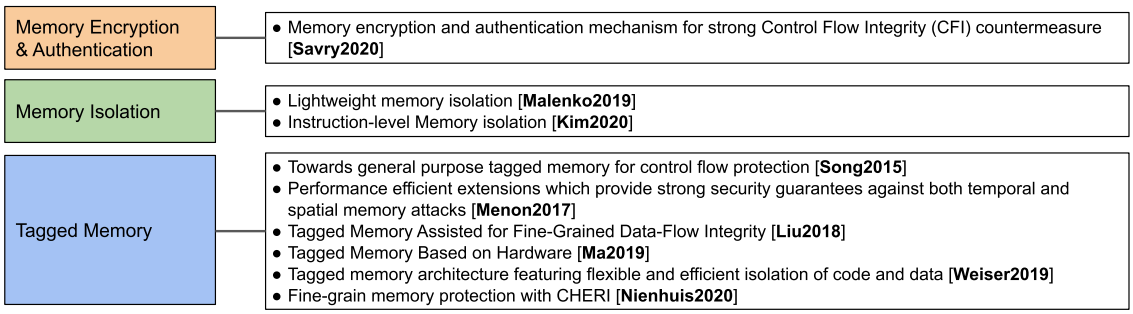


Fig. 7. RISC-V Memory Protection. *References:* ***Malenko2019****[69],* ***Kim2020****[54],* ***Savry2020****[70],* ***Song2015****[56],* ***Menon2017****[59],* ***Liu2018****[61],* ***Ma2019****[57],* ***Weiser2019****[71],* ***Nienhuis2020****[72]*

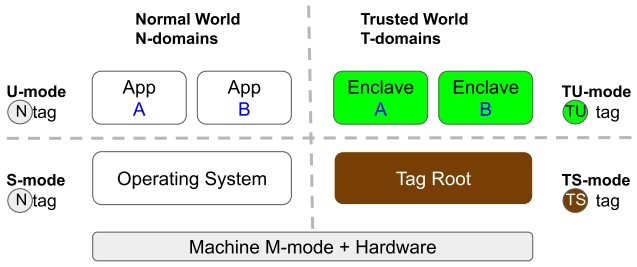
process of design, development, testing, and debugging. This methodology can satisfy many areas of the computer industry, but they fundamentally leave a large number of exploitable vulnerabilities, leading to many serious system security problems [72]. Effective memory protection can alleviate many system security vulnerabilities. As summarized in Figure 7, At least three types of memory protection solutions have been proposed: tagged memory[56, 57, 59, 61, 71, 72], memory isolation [54, 69], and memory encryption and authentication[70]. In this chapter, we discuss related research work on these topics.

4.1 Tagged Memory

The tagged memory [166, 167] enables a memory pointer to have a capability tag. Upon a memory access, the tag of the memory address will be checked to see if there is any capability violation. Tagged memory can prevent typical information leakage that may easily happen in classic programming languages. For example, the ISO C program can leak the secret key data because of an unintentional pointer access. Even worse, the compiler will not report any warning to this. With tagged memory such as its implementation in the Cheri C language extension [168], this kind of bug cannot happen because the capability checking mechanism will block unsafe memory access. Memory protection features have also been supported in some emerging programming languages such as the RUST, which is getting popular because of its security features and efficiency. Tagged memory is an active research domain of RISC-V hardware and architecture.

Song et al. [56] explored the performance of tagged memory by extending the Rocket RISC-V implementation [160] with preliminary tagged memory support. The implementation adds a predefined number of tag bits to each 64-bit word in memory. These tag bits are copied along with the data word through the cache hierarchy, meaning that each word in the L1 data and L2 cache lines are augmented with additional tag bits. The coherence of tags is maintained by the existing cache coherence mechanisms. Two new instructions *LTAG* and *STAG* are added for loading and storing tags. The tags are stored in a reserved memory area. Each access to a memory word also needs access to the tag, resulting in a memory traffic ratio of 2. Employing a tag cache can reduce the memory traffic ratio, and the traffic reduction is tag cache size dependent. Increasing the cache size from 16KB to 128KB can reduce the average memory traffic ratio from 1.59 to 1.06. Data-oriented attacks [169] manipulate non-control data to alter a program's benign behavior without violating its control flow integrity. It has been shown that such attacks can cause significant damage even in the presence of control-flow defense mechanisms. Based on the tagged memory feature, Liu [61] and Ma [57] et al. present tagged memory supported data-flow integrity mechanisms to enable fine-grained data-flow integrity checking to mitigate data-oriented attacks.

Many memory vulnerabilities are related to pointers [170]. Spatial memory attacks occur when a particular pointer accesses memory regions beyond its permissible range. Temporal memory attacks, on the other hand, occur when accessing a memory region that has been freed after allocation. Tagged memory is an effective scheme for memory pointer protection. Menon et al. [59] propose on-chip hardware design extension called the Base-and-Bound Cache (BnBCache), which optimizes fat-pointer performance via reducing the total number of memory accesses. It assumes that each 64-bit word in the memory is associated with a single Tag-Bit indicating whether



(a) Four security domains.

	N-tag	TC-tag	TU-tag	TS-tag
N-domains	✓	✘	✘	✘
TU-mode	✓	✘	✓	✘
TS-mode	✓	✓	✓	✓
M-mode	✓	✓	✓	✓

(b) Tag update permission in each security domain.

Fig. 8. TIMBER-V Fine-grained Enclaves based on Tag-Isolated Memory. This figure is adopted from paper [71].

the word is a pointer or regular data. The Tag-bits are set by the compiler and stored alongside the memory word even when it is loaded into a register, which also supports Tag-Bit. *BnBCache* consists of a *BnBIndex* table and a *BnBLookUp* table. The entries in the BnBIndex table have a 1 to 1 mapping with 32 general purpose registers. Each BnBIndex entry has an index pointing to a BnBLookUp entry, which consists of 4 fields: the base value (64-bits), the bound value (64-bits), the ptr_id (64- bits) and a valid bit. ISA extensions with eight new instructions are proposed to support the tagged memory mechanism. With the tag bit, valid bit, and boundary information, both spatial and temporal memory attacks can be blocked. The proposed solution, which is implemented on top of a RISC-V ISA based 64-bit baseline processor, incurs an area overhead of 1914 LUTs and 2197 flip flops on an FPGA without bringing critical path delay [59].

TIMBER-V [71] is a new tagged memory architecture. Combined with the memory protection mechanism, it can flexibly and efficiently isolate code and data to implement a Trusted Execution Environment (TEE) on small embedded systems. As Figure 8a demonstrates, execution in user mode (U-mode) and supervisor mode (S-mode) can both be separated in the normal world and the trusted world. The N-domains in the normal world support the traditional split between U-mode and S-mode, and allow existing code to run without modifications. Memory words in N-domains, no matter under U-mode or S-mode are encoded with the *N* tag. Memory words in T-domains under U-mode and S-mode are encoded with the *TU* and *TS* tags, respectively. Trusted user mode (TU-mode) can be leveraged for isolated execution environments, called enclaves. Trusted supervisor mode (TS-mode) allows to run TagRoot trust manager, augmenting the untrusted operating system with trusted services. Switching from N-domains to T-domains is implemented via trusted callable entry point functions, which is encoded with the *TC* tag. There are totally 4 different tags. Thus, TIMBER-V uses a two-bit tag per 32-bit memory word. Tags can only be updated within the same or a lower security domain but cannot be used to elevate privileges, as shown in Figure 8b. TS-mode (and M-mode) have full access to all tags. TU-mode can only change tags between N-tag and TU-tag to support dynamic interleaving of user memory. TU-mode is prevented from manipulating TC-tags, which are reserved for secure entry points. TIMBER-V uses the MPU to enhance the label isolation function to isolate each process while maintaining low memory overhead. TIMBER-V greatly reduces memory fragmentation and improves the dynamic reuse of untrusted memory across security boundaries. In addition to interleaving stacks, TIMBER-V can also implement novel execution stack sharing across different security domains. TIMBER-V is compatible with existing code, supports real-time constraints. A proof-of-concept implementation of TIMBER-V has been evaluated on the RISC-V simulator. Similar to the TIMBER-V, CHERI architecture also provides hardware capabilities that supports fine-grained memory protection and scalable secure compartmentalisation. Porting the tagged memory feature of CHERI to the RISC-V is on-going [72].

4.2 Memory Isolation, Encryption and Authentication

Existing memory isolation mechanisms suffer from scalability and performance issues, Kim et al. [54] propose instruction-level memory isolation (RIMI) to boost performance. RIMI also introduces the concept of domain to separate code and data in different memory map areas. The domain memory protection (DMP) mechanism only allows domain specific instructions to access instruction and data in a corresponding domain to achieve instruction-level memory isolation of different domain accesses. Each domain also consists of physical memory

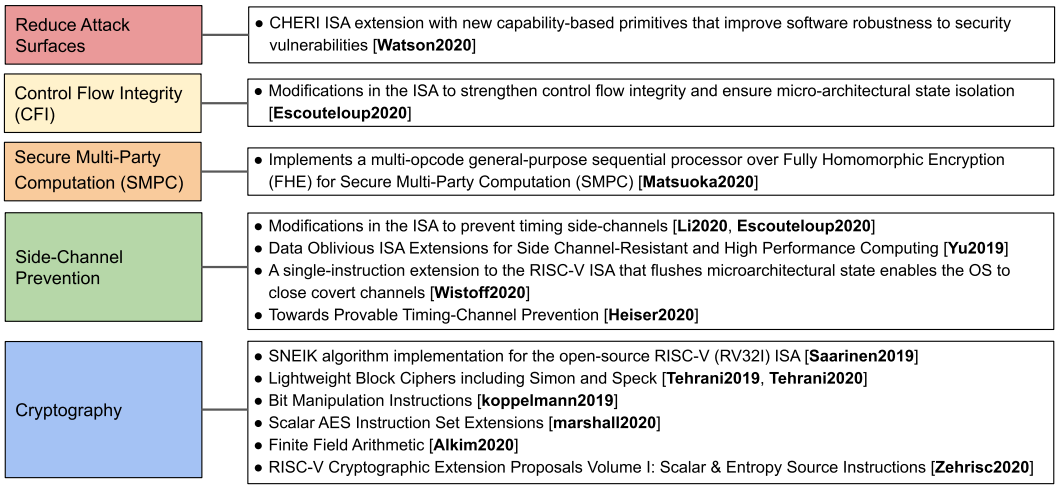


Fig. 9. RISC-V ISA Extensions for Various Security Objectives. *References:* ***Li2020****[73],* ***Escouteloup2020****[74],* ***Yu2019****[75],* ***Wistoff2020****[66],* ***Heiser2020****[67],* ***Watson2020****[76],* ***Saarinen2019****[77],* ***Tehrani2019****[78],* ***Tehrani2020****[79] ,* ***Koppelmann2019****[80],* ***Alkim2020****[81],* ***Marshall2020****[82],* ***Zehrisc2020****[83],* ***Matsuoka2020****[84].*

protection (PMP) regions and dedicated instructions to access its PMP regions. PMP and DMP configurations are jointly checked to determine the access permission of instruction and data. Specifically, RIMI implements memory *load* and *store* instructions with *domain_id* tags, and special control transfer instructions with *x* tags for domain switching. For example, *lw1* indicates the memory load instruction to access domain1, *sw0* indicates the memory storage instruction to access domain0, *jalr* can only jump between the same domain, and *jalrx* can jump between different domains. Evaluation based on the Spike simulator shows that using RIMI can effectively achieve shadow stack and in-process isolation. Also at the instruction level, Savry et al. [70] proposed a framework to ensure control flow integrity by implementing a lightweight masking scheme applied to instructions, which is based on an authenticated memory encryption mechanism. At the system level, for memory isolation, Malenko et al. [69] implemented a device driver isolation module in RTOS to prevent defective device drivers from destroying the state of the operating system and applications.

5 ISA SECURITY EXTENSIONS

The instruction set architecture (ISA) unifies the behavior of machine code running on the different CPU implementations of the ISA. The life cycle of ISA spans decades, but applications usually evolve quickly. In order to adapt to new application requirements, ISA may need to add new features to optimize performance, energy efficiency, or security. The instruction set extension is usually used to achieve the above goals while maintaining backward compatibility. Well-known instruction set extensions include x86 FPU, SSE, AVX, AES, SGX, and Arm SVE, Thumb, Neon, VFPv4 and TrustZone security extensions. As summarized in the figure 9, related extensions of RISC-V ISA have been proposed to enable hardware cryptographic functions [77, 79–83], reduce system attack surfaces [76], resist certain side-channel attacks [66, 67, 73–75], strengthen control flow integrity [74], and achieve secure multi-party computation [84]. We discuss RISC-V ISA security extensions in this Section.

5.1 Cryptographic Algorithms

Cryptographic algorithms are ubiquitous in platform security mechanisms. These algorithms are compute-intensive. There exist cryptographic standards and guidelines for a wide range and cryptographic functions including block cipher techniques, digital signatures, hash functions, and key management etc. Performance requirements and standard implementations of cryptographic functions make it a proper solution to integrate these functions as ISA extensions and implement them in hardware. We have demonstrated Armv8 Cryptographic

Extension for *AESE* and *AESD* instructions in Section 2.3.3. The RISC-V community is also extending the instruction set for similar purposes.

Secure and efficient implementation of AES is a basic requirement on most computing platforms. Therefore, dedicated instruction set extensions (ISEs) are often implemented to support efficient AES execution. RISC-V is a new ISA and lacks this standardized ISE. Marshall et al. [82] investigated the latest industrial and academic ISEs for AES, and evaluated five different ISEs. Compared with the software T-table based implementation, ISEs for 32-bit and 64-bit architecture can achieve 4× and 10× AES-128 block encryption performance improvement with hardware costs of 1.1K and 8.2K gates, respectively. They also explored how to use RISC-V standard bit manipulation extension [171] to effectively implement AES-GCM. Their work is part of the ongoing RISC-V cryptography extension standardization process. RISC-V Cryptographic Extensions Task Group are working on cryptographic extension proposals for scalar and entropy source instructions, which covers bit manipulation, scalar AES, SHA, SM3 and SM4 acceleration, and TRNG entropy source interface [83].

Similar to the AES GCM algorithm, SNEIK [172] is a lightweight, permutation-based encryption primitive code library that can perform cryptographic hashing, authenticated encryption with associated data, and other tasks. The design is designed to meet all symmetric cryptographic needs, including tasks such as pseudo-random number generation and key derivation. Saarinen et al. [77] evaluated SNEIK on RISC-V (RV32I) and showed that SNEIKEN128 can conduct authenticated encryption at 54.8 instructions/byte, which roughly matches the performance of AES-128 on the comparable Arm platform. SNEIKHA256 achieves 98.6 instructions/byte on RV32I. The RV32I base instruction set lacks rotation instructions, which results in lower throughput than Armv7. They observed that the structure of SNEIK permutation was very suitable for ISA extension optimization. RV32I extension only has an impact of 258 LUT / 65 slices on FPGA resource utilization, but it increases the SNEIK permutation speed by 7×. Tests conducted on Artix-7 FPGA hardware showed that the RISC-V "Crimson Puppy" SoC with ISA extension can perform SNEIKEN128 operation at 12.4 cycles/byte, and SNEIKHA256 operation at 17.3 cycles/byte, demonstrating that a simple RISC-V instruction set extension can achieve 5× acceleration.

Tehrani et al. [78, 79] provides a detailed architecture and implementation of specific processor instructions for lightweight encryption algorithms. These instructions target the 32-bit RISC-V ISA and allow acceleration of several commonly used lightweight block ciphers. They used the plug-in-based architecture of the VexRiscv processor to implement instruction extension on the Artix-7 FPGA board. They demonstrated the hardware resource usage of these extended instructions. For a representative lightweight block cipher, they compared the performance of the system with ISA extensions and the base system. The results show that the instruction extension can accelerate the lightweight encryption algorithm by 33 to 138× at a reasonable hardware cost. We will further discuss lightweight cryptographic algorithms in Section 6.

Bit manipulation is the act of processing bits shorter than words. Cryptographic algorithms require a large number of bit operations, so the support of bit operations has a significant impact on the performance of cryptographic algorithms. Koppelmann et al. [80] proposed RISC-V extensions for bit manipulation instructions (BMIs). Specifically, they extended the RISC-V ISA with ten bit manipulation instructions: parity, byte swap, right/left rotation, popcount, bit reverse, count leading/trailing zeros, and parallel gather/scatter. These BMIs achieve the same functions as the current x86 BMIs, while the required code bytes are reduced by 13.5%. In order to prove its efficiency, they evaluated the extensions using 13 benchmarks, and the new instructions showed good acceleration. Wolf et al. [171] in RISC-V BitManip Task Group is actively working on the bit manipulation instruction extensions. According to the RISC-V Bitmanip Extension Document Version 0.94-draft [171], they have proposed more than 100 bit manipulation instructions for both 32 and 64-bit ISAs, covering bit manipulating, bit permutation, bit field place, bit compress/decompress etc.

The development of quantum computing and the devastating impact of Shor's algorithm on our current IT security has spawned active research on encryption systems that prevent quantum computing attacks. This field of research is called post-quantum cryptography (PQC). An important aspect of PQC research is the effective and secure implementation of PQC algorithms. Current cryptographic algorithms require effective arithmetic operations on hundreds to thousands of bits of data, while many PQC schemes perform finite field operations on data less than 20 bits. Alkim et al. [81] take the lattice-based key encapsulation mechanisms Kyber and NewHope as examples to study the impact of providing ISA extensions with finite field operation support on

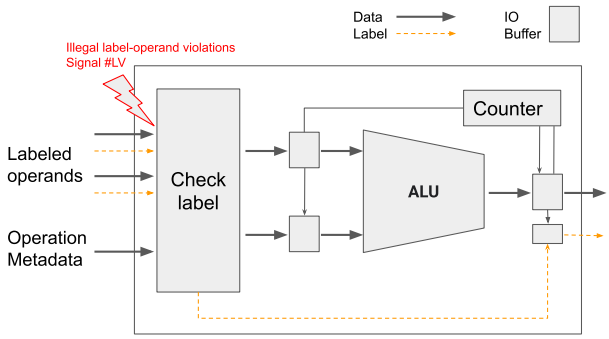


Fig. 10. Label station for an execution unit with one internal arithmetic unit in OISA architecture [75].

the performance of the PQC schemes. They create a prototype implementation of the presented instructions on the VexRiscv core, and evaluate the design on two different FPGA platforms. The result shows a speedup of polynomial arithmetic of up to 85% over the basic software implementation. The custom instructions can replace a general purpose multiplier to achieve very compact implementations.

Fault attacks and power analysis threaten the implementation of cryptographic schemes. Shielded power analysis and redundancy-based methods are the most commonly used countermeasures for this type of attack. With these attacks in mind, NIST recently requested the submission of documents to illustrate the possibility of adding countermeasures against these attacks at low cost. Steinger et al. [94] propose an instruction extension of Ascon-p, which uses tight integration with the processor register file to significantly accelerate various symmetric encryption calculations at a relatively low cost. As a proof of concept, they integrated the instruction extension into the 32-bit RI5CY core. They evaluated various hardware indicators and showed that the accelerator can be implemented with about 4.7 kGE, in other words about half the area of the dedicated coprocessor design. Considering this built-in acceleration of Ascon-p, they created an assembly version of the Ascon/Isap model, which utilizes instruction extensions and provides benchmarks for authentication encryption, hashing, and pseudo-random number generation. Based on the benchmarks, compared with the pure software implementation, instruction extensions of Ascon-p achieved a speedup of 50 to 80.

5.2 Side-channel Attack Prevention

As we discussed in Section 3.1, preventing microarchitecture side-channel attacks is one of the most pressing challenges in hardware security today. Meltdown[20] and Spectre[21] belong to this type of attack. In order to prevent side-channel attacks, hardware-based technology focuses on redesigning the cache and completely modifying the processor architecture to improve cross-processor information flow tracking. The software-based method recommends clearing all core states, including private caches, translation backup buffers, branch prediction units, etc. The software-based method relies heavily on the refresh mechanism. Although refreshing or clearing the persistent state of multiple vulnerable hardware components at the core level (within level 1) is essential to create good and complete time isolation in the entire system, research shows that the existing ISA refresh instructions are functionally incomplete and inefficient in performance and power efficiency.

There has been related work trying to prevent side-channel information leakage by obliviously writing program data. In this model, program writing needs to avoid sensitive data access leaving traces on shared resources. Despite recent efforts, the security and performance of running data-oblivious programs on modern computers are questionable. First, writing a data-oblivious program assumes that certain instructions in the ISA will not leak information, but the ISA and hardware do not provide such guarantees. Second, writing programs to avoid data-dependent behavior will inevitably incur serious performance overhead. Utilizing ISA extension to prevent side-channel attacks has been proposed [66, 67, 73–75]. In this section, we discuss in detail the related work of preventing side-channel attacks through RISC-V instruction extension.

Yu et al. [75] proposed a data oblivious ISA extension (OISA) for RISC-V. In terms of security, the proposed ISA design can block side channels. In terms of performance, the OISA supports effective memory oblivious

calculations, and has security features that can remain enabled in common situations such as out-of-order speculative execution and other modern hardware optimizations. Based on the RISC-V out-of-order, speculative BOOM processor, they implemented a complete hardware prototype. Through formal analysis of an abstract BOOM-style machine, they proved that OISA can achieve its security design goals. They evaluated the area overhead of the hardware mechanism, and provided performance experiments to show how OISA can improve various existing data oblivious codes, including constant-time cryptography and memory oblivious data structures, while also improving their security and portability. The label station is the OISA core component. As Figure 10 demonstrates, the label station checks and tracks Public/Confidential labels as data flows through the pipeline and signals *#LV* when violations occur. The result label, which travels with the result and accompanies the entire life cycle of the result, is computed based on operand labels.

We have discussed timing channels in Section 3.1. ISA extension is also a potential way to prevent timing channels. Li et al. [73] created a dedicated flushing instruction to improve the efficiency of temporary isolation at the core level to mitigate the possibility of potential timing channels. They first propose a single instruction multiple refresh (SIMF) scheme, which integrates the refresh operation in one instruction to clear the core-level state. The main advantages of SIMF are: 1) It greatly reduces the dynamic instruction count dedicated to refresh (resulting in cycle counting and instruction fetching capabilities); 2) It requires minimal expansion of existing hardware (adding an instruction to the ISA); 3) When SIMF is not in use, in the case of explicit barrier instructions, the sequence of refresh operations is implicitly executed in an instruction; 4) It brings benefits for programming, including atomicity and simplicity. They prototyped SIMF in an open source scalar ordered RISC-V processor. They extended the RISC-V ISA with another instruction called *FLUSHX*, which refreshes the core-level state, including L1/L2 TLB, L1 cache, and branch prediction units (BTB, RAS, BHT). Wistoff et al. [66] proposed a similar solution targeting the similar timing channel issue. Specifically, they propose a new RISC-V fence instruction with arguments to enable the operating system to control state flushing. The evaluation shows that the proposed scheme completely eliminates the timing channels including L1 data and instruction cache channels, TLB, branch target buffer (BTB), and branch history table (BHT) channels.

5.3 SMPC and CFI

The goal of secure multi-party computation (SMPC) is to create methods for all parties to jointly compute functions on their inputs, while keeping those inputs private [120]. Traditionally, cryptography is about hiding content, and this new type of calculation and protocol is about hiding part of the information about the data while using data from multiple sources for calculations and correctly generating output. By customizing RISC-V ISA, Matsuoka et al. [84] have proposed the Virtual Security Platform (VSP), which is a comprehensive platform that can provide a complete set of tools for a complete two-party secure computation offloading (SCO) solution. The VSP includes open source design and implementation of homomorphic encryption library, processor architecture, custom ISA and compiler environments. Based on the famous Torus Fully Homomorphic Encryption (TFHE) scheme, VSP allows any user with any C program to execute its code in SCO mode.

The goal of Control Flow Integrity (CFI) [173] is to prevent malware from attacking the control flow of the redirector. We have discussed hardware-assisted security units to achieve CFI in Section 3.2. CFI can also be enforced through ISA extensions. Escouteloup et al. [74] discuss changes to the ISA to strengthen CFI and ensure the isolation of microstructure states. They put forward some security recommendations in the ISA design, such as marking certain registers such as the frame pointer as confidential to disable branching on confidential registers, authorizing instructions only when the timing of the instruction is not data-dependent, and prohibiting forward indirection jump and prohibit all micro-architectural management instructions, especially cache management, and must provide micro-architectural security guarantees through hardware security context (HSC) instructions.

5.4 Reduce Attack Surface

The attack surface of the system is defined as the attackability of the system in the three abstract dimensions of method, data and channel [174]. Intuitively, the larger the attack surface, the more likely the system is to be attacked, and therefore the less secure it is. Capability Hardware Enhanced RISC Instructions (CHERI) [76] extends

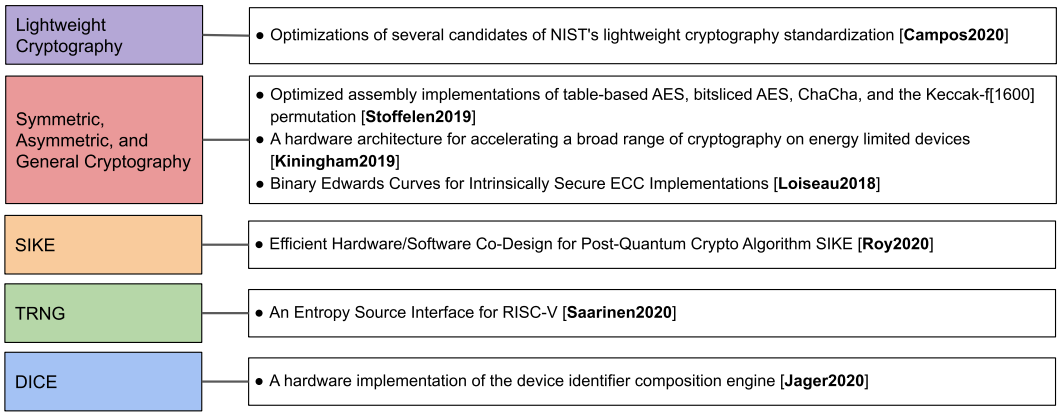


Fig. 11. Cryptographic Primitives. *References:* ***Stoffelen2019****[98]*, ***Kiningham2019****[88]*, ***Campos2020****[97]*, ***Roy2020****[96]*, ***Saarinen2020****[87]*, ***Jager2020****[90]*, ***Loiseau2018****[91]*.

Table 3. Performance of Polarfire RISC-V FPGA vs. Xeon x86 Data Center-level CPU Cores. The evaluation is based on openssl cryptography implementation with a single thread. The metrics are KB per second processed for HMAC, SHA, and AES, operations per second for ECDH, and sign or verify operations per second for RSA and ECDSA.

	HMAC (MD5)	SHA256	AES-256 CBC	ECDH	RSA		ECDSA	
					Sign	Verify	Sign	Verify
Polarfire RISC-V FPGA (660MHz)	32421	7392	6081	155	3693	54.4	84	57
Xeon x86 Server (2.5GHz)	572497	386888	173803	11576	176056	45995	15028	20085
Performance Ratio	5.66%	1.91%	3.50%	1.34%	2.10%	0.12%	0.56%	0.28%

the instruction set architecture (ISA) with new function-based primitives, thereby improving the robustness of the software in terms of security. The CHERI model follows the principle of least privilege, and achieves higher security by minimizing the privileges that can be accessed by running software. Another guiding principle that CHERI adheres to is the principle of intentional use, that is, when a software can use many privileges, the use of privileges should be clearly specified, rather than an implicit choice. CHERI reduces the attack surface of the system, and even if an attacker successfully exploits the vulnerability, they will gain the least permissions. CHERI has previously been applied to MIPS and Arm ISAs. Recently, CHERI and its complete software stack has been ported to the RISC-V 32-bit and 64-bit variants [76]. CHERI-RISC-V specification shares a lot of architectural features with the CHERI-MIPS such as the tagged memory. But they have different interpretations of addresses in memory capabilities. Also, CHERI-related page permissions are added to RISC-V architectural page-table formats instead of the MIPS translation lookaside buffer (TLB) entries.

6 CRYPTOGRAPHIC PRIMITIVES

Trusted computing [175], communication [176], storage [177], execution [50], and many other security objectives rely on cryptographic primitives including symmetric and asymmetric ciphers, data integrity algorithms, mutual trust key management and distribution mechanisms. As the application ecosystem evolves, new challenges and requirements of cryptographic primitives emerge. Figure 11 summarizes the cryptographic primitive implementation or performance optimization projects in the RISC-V domain.

6.1 Lightweight Cryptography

First, small IoT devices require lightweight cryptographic algorithms. It is common that small IoT devices are CPU or memory resource-constrained, they may not be powerful enough to support efficient execution of standard cryptographic algorithms. As Table 3 shows, the latest Polarfire RISC-V FPGA has an average single-core compute performance of only 3.1% of the Intel Xeon 4215 x86 processor. Lightweight cryptography has been proposed to meet the requirements of resource constrained devices [97]. Lightweight conceptually can refer to chip area size, code/memory size, energy efficiency, etc. There is an ISO/IEC 29192 standard for lightweight block ciphers. CLEFIA [178], PRESENT [179], LEA [180] are three of the ISO block cipher algorithms. NIST is also screening lightweight cryptography algorithms, which will be included into the NIST lightweight cryptographic standard. A major challenge of applying encryption in restricted environments is the trade-off between security and performance. Fabio et al. [97] analyze different strategies for optimizing candidate solutions of the NIST lightweight cryptographic algorithms on the RISC-V architecture. Specifically, they demonstrated how multiple lightweight NIST candidates such as Gimli, Sparkle, Saturnin, Ascon, Delirium, and Xoodyak can be efficiently implemented. They studied the overall impact of optimizing symmetric key algorithms in assembly and C languages, proposing optimizations such as loop unrolling that can speed up the software-implemented algorithms by up to 81%.

6.2 Symmetric and Asymmetric Cryptography

Stoffelen et al. [98] highlight the features of RISC-V, and provide optimized assembly implementation of table-based AES, bit-sliced AES, ChaCha and Keccak-f[1600] for the RV32I instruction set. Regarding public key cryptography, they study the performance of arbitrary-precision integer arithmetic without the carry flag. They conduct quantitative performance studies on several RISC-V extensions, which provide design insights for future RISC-V core design and implementation.

Fixed function hardware accelerators such as an AES engine cannot support new ciphers. Kiningham et al. [88] introduce Falcon, a hardware architecture used to accelerate various ciphers on energy-constrained devices. Falcon provides a general execution engine that support bitslice and permutation instructions, which are the backbone operations of current and probably future dominant ciphers including AES, Cha-Cha, SHA-256, RSA, Curve25519 ECC, and post-quantum cryptography such as R-LWE. For encryption technology, Falcon provides software flexibility while reducing the energy consumption of ciphers by 5 to 60× compared with software implementations. This improvement makes it feasible for IoT applications to upgrade the ciphers after deployment, so that they can always stay abreast of the latest security practices without shortening device deployment life or sacrificing application workload.

To deal with the security challenges of IoT devices, Loiseau et al. [91] propose a fast, low-power encryption technology for a new set of Binary Edwards Curves that have been defined to reach up to 284-bit security level suitable for IoT devices embedded with 32-bit general-purpose processors. They optimized the choice of point generators using w coordinates to save multiplication in addition and doubling formulas. They managed to calculate one step of the Montgomery ladder with 4 multiplications and 4 squares. In addition to performance advantages, encryption on this curve also has inherent security properties against physical attacks.

6.3 SIKE (Post-quantum Cryptography)

Advances in quantum computers put the security of public key cryptosystems into risk [181]. The security foundation of public key cryptosystems is that the integer-factorization problem is very complicated when the integer is very large. However, the quantum computer can solve this problem exponentially faster than traditional computers, which puts the existing public key cryptosystems at risk. Thus, if quantum computers are realized, public key cryptosystems may be easily compromised. In this background, NIST is screening public key encryption and digital signature algorithms that can resist potential attacks of quantum computers. The post-quantum algorithms will augment FIPS 186-4, SP 800-56A, SP 800-56B. NIST has selected 7 finalists and 8 alternates as candidates in their round 3 screening. The algorithm evaluation will be finalized very soon.

SIKE public-key encryption and key-establishment algorithm is one of the NIST alternate candidates [182]. It is a promising candidate standard, but its algorithms are resource-intensive. Although SIKE's FPGA implementation provides low latency and high performance, it has the disadvantages of large area and low flexibility. Compared with FPGA implementation, pure software implementation has much lower performance.

Software and hardware co-design are important to optimize performance and satisfy design constraints such as cost and power consumption, as well as to shorten the time to market considerably [183]. The openness of RISC-V provides unprecedented hardware optimization opportunities. Roy et al. [96] propose the hardware/software co-design method of SIKE, integrating the finite field accelerator based on redundant numbers into two microcontroller platforms based on Arm and RISC-V. The results show that, compared with the independent software implementation on Arm32 and Arm64, the implementation on the Arm Cortex-A9 enhanced with the field accelerator provides a significant speedup in terms of clock cycles. In addition, in order to show how to reduce the communication overhead between the processor and the accelerator, they directly integrate the finite field accelerator into the core of the RISC-V processor. This is the first hardware and software co-design method to implement SIKE design on Arm and RISC-V platforms. The proposed design requires 65500K clock cycles to execute SIKE on the Arm Cortex-A9 processor. On RISC-V, the proposed design only requires 36900K clock cycles.

6.4 TRNG

As we have discussed in Section 2.2, the random number generator is an important root of trust module. Saarinen et al. [87] propose RISC-V true random number generator (TRNG) architecture that separates the entropy source component and the encrypted PRNG into a single interface. This is different from the previous TRNG implementations. They describe the interface and its use in cryptography, and discuss the background and basic principles of the interface. The design refers to the mainstream ISA, the latest SP 800-90B and FIPS 140-3 entropy review requirements, AIS-31 and Common Criteria for IT Security Evaluation, as well as current and emerging encryption requirements such as post-quantum encryption. The architecture choice is the result of quantitative observations on secure microcontrollers, Linux kernels and random number generators in cryptographic libraries. They further compare this architecture with some contemporary random number generators and describe a minimal TRNG reference implementation using an entropy source and RISC-V AES instructions.

6.5 DICE

The Device Identifier Composition Engine (DICE) is the minimal requirement for trusted computing on microcontrollers. Currently, most implementations use hardware that is not specifically designed for this purpose. These implementations rely on black-box MPUs. Because hardware that is not originally designed for DICE is used, there are certain pitfalls in the implementation process. Jager et al. [90] propose a DICE architecture based on a microcontroller, which is equipped with hardware that meets DICE requirements. It includes minor modifications to the processor pipeline, dedicated memory blocks, and modified interrupt and debug modules. They create an FPGA prototype based on the VexRiscV platform and evaluate the impact of the increase in chip size and DICE extension on the runtime to prove that DICE can be implemented with minimal changes to the microcontroller design and used in IoT and automotive environments as a trusted component.

7 PROTECTION AGAINST SIDE-CHANNEL ATTACKS

Side-channel attacks can be carried out through power analysis [184] and electromagnetic analysis (EMA) [185]. In Section 5.2, we discussed the research on preventing side-channel attacks through ISA extensions. In addition to ISA extensions, there are many general methods such as implementing countermeasures in the CPU execution pipeline, or checking whether the hardware is in constant execution by analyzing the RTL code of the circuit. In this section, we discuss some general methods to detect and prevent side-channel attacks. Specifically, as Figure 12 shows, we will discuss the branch prediction, TLB, inter-core, and timing side channels, and discuss the power analysis and electromagnetic attacks that exploit side channels. We will also discuss virtual prototyping, which can detect simple side channels.

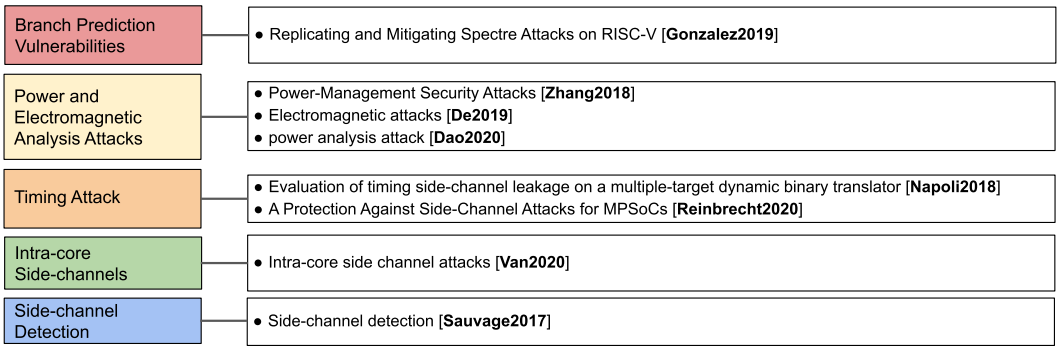


Fig. 12. Protection against Side-channel Attacks. *References:* ***Deng2019****[99]*, ***Gonzalez****[100]*, ***Napoli2018****[101]*, ***Reinbrecht****[102]*, ***De2019****[103]*, ***Zhang2018****[104]*, ***Dao2020****[105]*, ***Van2020****[106]*, ***Sauvage2017****[107]*.

7.1 Branch Prediction Vulnerabilities

The side-channel vulnerabilities of modern processors make hardware security a top priority in processor design. Gonzalez et al. [100] demonstrated how a general-purpose open source Berkeley Out-of-Order Machine (BOOM) based on a RISC-V processor can be used to research mitigating side-channel attacks at the microarchitecture level. First, they replicate several basic variants of the Spectre vulnerability [21], which utilize speculative execution in the L1 data cache. Then, they implement preliminary hardware mitigation measures against this type of attack, prove its effectiveness, and measure its impact on performance and area size. Compared with the baseline processor, evaluation of the hardware mitigation shows that in the 45nm process, the IPC has increased by 2%, the area has increased by 2.5%, and the clock has been reduced by 0.36%. This work confirms the value of the open source RISC-V hardware ecosystem for secure hardware research.

7.2 Power and Electromagnetic Analysis Attacks

Running unprotected software on an unprotected microprocessor can cause various side-channel leakages including direct-value leakage, data-overwrite, and circuit-level leakage [103]. The software implementation of cryptographic algorithms is vulnerable to side-channel analysis (SCA) attacks because the cryptographic key may leak through the processor's measurable physical characteristics such as power consumption and electromagnetic radiation. Various algorithms with solutions have been proposed to mitigate this issue. However, they rely on equipment assumptions that are rarely met, thus are not easy to implement. Moreover, these solutions do not consider microarchitecture-related issues. Mulder et al. [103] propose integrating the countermeasures of side-channel analysis into the RISC-V implementation. They use masking technology to wrap the secret before writing it to memory and do a reverse operation to unwrap the secret after reading it back from memory, so as to protect memory access from SCA. The solution prevents first-order power or electromagnetic attacks while keeping the implementation cost as low as possible. The evaluation results confirm the security of various encryption primitives running on the protected hardware platform.

Dynamic Frequency Scaling (DFS) is a technology related to the dynamic change of the clock frequency and usually the associated voltage of a CPU or hardware module during operation to adapt its power consumption [186]. Dao et al. [105] demonstrates how to integrate DFS technology into an open source RISC-V processor and use it as a simple, cost-effective countermeasure against Simple Power Analysis attacks. The idea is that the DFS module can conceal sensitive information in the measured power trace, while the processor's hardware resource requirements hardly change. They implemented DFS in the Sakura-X FPGA board and demonstrated the usefulness of this method to mitigate Simple Power Analysis attacks.

Most modern computing devices can control frequency and voltage through fine-grained operations. CLKSCREW [187] is a new type of fault attack, which uses the security obliviousness of the energy management mechanism to make the frequency of the equipment exceed its operating limit, which leads to malfunction and security violation. Statically and permanently limiting the frequency and voltage modulation space can mitigate such

attacks, but will result in a significant drop in performance and energy efficiency. Zhang et al. [104] propose a runtime technology that uses a neural network model to dynamically blacklist unsafe operating performance points. The model is first trained offline at design time, and then adjusted at runtime by checking a set of selected functions such as power management control registers, timing error signals, and core temperature. They designed an algorithm and hardware called the BlackList (BL) core, which can detect and mitigate this kind of power management-based security attacks with high accuracy. The BL core generates a relatively small amount of overhead in terms of power, delay, and area size.

7.3 Timing Attack

The calculation of algorithms takes time. If the algorithm is not carefully designed with constant execution time, it may cause different inputs to have different execution times. If the calculation involves sensitive information such as a cryptographic key, the attacker may reverse the content of the key by traversing a large number of input vectors and accurately measuring the execution time of each test vector, causing the leakage of sensitive information. Prying sensitive key data through time information is usually much easier than cryptanalysis using known pairs of plaintext and ciphertext.

The Translation Lookaside Buffers (TLBs) in modern processors may cause timing side-channel attacks. But exploiting the TLB channel is challenging due to unknown addressing functions inside the TLB and the attacker's limited monitoring capabilities that at best cover only the victim's coarse-grained data accesses. However, recently researchers can reverse engineer the addressing functions inside the TLB, and devise a machine learning strategy that exploits high-resolution temporal features about a victim's memory activity, which make TLB side-channel attack practical [188]. To mitigate this risk, Deng et al. [99] introduce a novel three-step modeling method that is used to exhaustively enumerate all possible TLB-based timing vulnerabilities. Step 1 performs memory operations and places the TLB block in a known initial state. Then, step 2 performs the second memory operation to change the state of the TLB block. Finally, Step 3 performs the final memory operation, and the time of the final operation depends on the addresses of Step 1, Step 2 and Step 3. Attacks with more than three steps can be broken down into a three-step attack. Based on the three-step model, they show how to automatically generate a micro-security benchmark that can test TLB vulnerabilities. They propose two new secure TLB designs: static partition (SP) TLB and random fill (RF) TLB. The evaluation of the secure TLB implemented in RISC-V Rocket Core shows that the new TLB can not only defend against previously announced attacks, but also against other new timing-based attacks in the TLB discovered using the new three-step model. The FPGA-based evaluation shows that the RF TLB defends against all attacks with performance overhead of less than 10%.

Multi-processor system-on-chip (MPSoC) is a popular computing platform suitable for various applications due to its energy efficiency and flexibility. SoCs with heterogeneous architecture allow for the integration of various central processing units and even graphics processors on the same system are getting popular [189]. Like many other platforms, they are also vulnerable to side channel attacks (SCA). Logical SCA can retrieve sensitive information by simply observing the system attributes that depend on the software executed by the victim on the MPSoC, which is very harmful. Unfortunately, many current protection mechanisms are either platform-dependent or only effective against a few attacks. Reinbrecht [102] introduces Guard-NoC, which is a secure network-on-chip (NoC) architecture that can protect MPSoC from various Logical SCAs. The secure NoC uses three application-independent strategies to hide and isolate sensitive information by masking the execution time of the operation and employing dual communication strategy such as using packet and circuit switching at the same time. Packet switching is used for secure packets and circuit switching is used for common packets. Evaluation shows that the security NoC can resist the actual Logical SCAs, and hardly leak any information, while having a minimal area size and power consumption.

Timing side channel attacks are an important issue in cryptographic algorithms. If the execution time of the implementation depends on secret information, the adversary can infer the secret by measuring the execution time. Different methods have recently emerged to explore information leaks in encryption implementations and protect them from these attacks. For example, in Section 3.1 we have discussed IODINE [64], which translates Verilog code for a formal analysis to detect timing channels. However, there is very little about ISA emulation

and its impact on timing attacks. Napoli et al. [101] studied the impact of OI-DBT, a dynamic binary translator using different region formation technologies (RFT), on the implementation of constant time and non-constant time of cryptographic algorithms. Experiments show that simulation can have a significant impact on secret leakage, and even mitigate leakage in some cases. In addition, the results show that the simulator's choice of RFT heuristic also has an impact on these leakages.

In Section 3.2, we have discussed program obfuscation as a widely used intellectual property (IP) protection technique against reverse engineering attacks. Biswas et al. [190] observed that the choice of transformation sequence has a significant impact on timing channel information leakage. Certain transformation sequences may cause leakage higher than the original program. Biswas et al. proposed a timing channel sensitive program obfuscation optimization framework based on the genetic algorithm to find the best combination of obfuscation transformation functions in terms of performance and prevention of timing side channel leakage. They evaluated the new framework on the RISC-V Rocket core. They use the *dudect* tool to verify that the proposed TSC-SPOOF framework provides optimization points for ModExp and MulMod16 programs, while reducing timing channel leakage. They observed that for the optimized ModExp and MulMod16 programs, it takes about 1M and 2M measurements, respectively, to pass through the t-statistic of 10. But for the two programs in the initial population, only 20K measurements are needed to cross the same t-statistic.

7.4 Intra-core Side-channels

Systems that protect the enclaves from privileged software must consider software-based side-channel attacks. Protection against attacks that can be launched from privileged software is an emerging attack model. Van et al. [106] propose to protect against intra-core side-channel attacks by enforcing that all active enclaves are physically isolated on their own separate core, thereby mitigating side channel attacks inside the core. They also redesign the memory hierarchy based on the ownership of the security zone to protect the security zone from intra-core side channel attacks. The combination of physical isolation and a redesigned memory hierarchy can protect the enclave from all known software-based side-channel attacks. The memory tag is used to protect the confidentiality and integrity of the enclave's memory. Bootstrapping Shim is the RoT, from which the management shim can start to manage the tags that protect the memory. The management shim is the software part of the TCB. The hardware forces enclaves to only access the pages they are authorized to access, and the management shim is the only code that is allowed to change the value in the tag directory. The management shim is not an operating system, it only implements the minimum logic required to securely maintain the enclave life cycle and transfer page ownership. They implemented the system and evaluated it with communication performance, memory overhead, and hardware area metrics. Evaluation shows that adding secure cores to a modern system on chip would increase CPU complexity by about 14.5%, and increase the hardware area by less than 2%. Storing the extra tag data increases the size of the last-level cache by 2% to 13%.

7.5 Side-channel Detection with Virtual Prototyping

Evaluating software security vulnerabilities in the design phase can find problems at the earliest stage and avoid the cost of later vulnerabilities repair, which is therefore very critical. Sauvage et al. [107] describe a virtual prototype implemented by scalar multiplication, aiming to protect a platform from simple side-channel attacks. They used the Mentor Graphics Modelsim tool to obtain a reproduction of information leakage as close to reality as possible, requiring bit and clock cycle accuracy to simulate the execution of the software implementation on PULPino [191], which is an open source 32-bit RISC-V microcomputer. For each clock cycle, they calculate the number of bits entering the microcontroller, the power consumption image, and observe the program counter to identify the executed assembly instructions, and then identify the corresponding C function. They use naive double-and-add implementation relying on cryptographic primitives of the mbed TLS library for reference. Virtual analysis points out that there are differences between the *double* function on one side and the *add* function on the other side in the way of managing variables and internal operations, which can be exploited to extract the private key. This method is still immature, and it faces many challenges to get practical applications, such as improving simulation performance, more realistic attack models, and more automated deployment.

8 CONCLUSION AND FUTURE WORK

Hardware-based security technologies such as memory protection and instruction set security extensions have been widely used in practice. Since the Meltdown and Spectre vulnerabilities broke out, the security of computer hardware and architecture has received extensive attention from industry and academia, attracting more and more research. As a new open instruction set, RISC-V has received extensive attention, and is moving towards the mainstream. Although there are more and more research projects related to RISC-V, the security research of RISC-V as a new architecture is relatively lagging. In this article, we investigate the current status of RISC-V hardware and architecture security research, hoping to provide readers with a big picture of the RISC-V security ecosystem. Specifically, we first briefly describe the background and architectural security foundation of RISC-V, and then introduce the security research of RISC-V by topic. We focus on discussing hardware and architecture security. Our research covers hardware and physical access security, hardware-assisted security units, ISA security extensions, memory protection, cryptographic primitives, and side-channel attack protection. During the investigation, we notice that the hardware and architecture foundation of RISC-V security is in place, and a wide range of security mechanisms have been established.

We also notice that the cryptographic instruction set of RISC-V is not complete, and the standardisation work is still in progress. The instruction set is the biggest space for improvement of RISC-V in the architecture layer. Realizing efficient cryptographic algorithms through the instruction set will be significant work. In addition, the security foundation of RISC-V architecture, such as PMP, may be attacked via side-channels [192]. Preventing the architectural security foundation from being attacked by the side-channels will be an important research topic. As we mentioned above, RISC-V is a new architecture, and some research is relatively lagging behind. Porting classic security applications of other architectures to RISC-V will be major and important tasks in the near future. For example, the RISC-V community is porting Arm's OP-TEE implementation. It will be meaningful to compare RISC-V with other architectures such as Arm in related fields. In addition, we can also see from our summary that although the current RISC-V research has covered a wide range of security topics, many researches are still in an early stage and related implementations are not yet mature. For example, the current research on memory protection is mainly focused on tagged memory, only a few researches are on memory isolation and encryption verification. Research on logic locking, electromagnetic injection attacks, side-channel prevention and detection, and control flow integrity is still immature. RISC-V security topics still have broad research space. In this article, we mainly discuss the hardware and architecture security of RISC-V. As the next step, we will continue to conduct a comprehensive study of RISC-V firmware and system, as well as software and application security, in order to achieve complete coverage of RISC-V security spectrum.

As far as we know, this is the first comprehensive review article on RISC-V security. We hope that this article will give readers a comprehensive understanding of existing RICS-V security mechanisms, and even from a broader perspective, understand the state-of-the-art research on embedded systems and general-purpose computer architecture security.

REFERENCES

[1] Alasdair Armstrong, Thomas Bauereiss, Brian Campbell, Alastair Reid, Kathryn E. Gray, Robert M. Norton, Prashanth Mundkur, Mark Wassell, Jon French, Christopher Pulte, Shaked Flur, Ian Stark, Neel Krishnaswami, and Peter Sewell. "ISA Semantics for ARMv8-a, RISC-v, and CHERI-MIPS". In: *Proc. ACM Program. Lang.* 3.POPL (Jan. 2019).

[2] Krste Asanović and David A Patterson. "Instruction sets should be free: The case for risc-v". In: *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2014-146* (2014).

[3] SiFive. *HiFive Boards*. https://www.sifive.com/boards. Online; accessed 10 December 2020.

[4] SiFive. *SiFive RISC-V cores for SSD Controllers*. https://sifive.cdn.prismic.io/sifive%2F83156596-8265-4613-837a-bcd7b564c080_sifive-risc-v-cores-for-ssd-controllers.pdf. Online; accessed 10 December 2020.

[5] Chen Chen, Xiaoyan Xiang, Chang Liu, Yunhai Shang, Ren Guo, Dongqi Liu, and Yimin Lu et al. "Xuantie-910: Innovating Cloud and Edge Computing by RISC-V". In: *2020 IEEE Hot Chips 32 Symposium (HCS)*. 2020, pp. 1–19.

[6] Daniel Petrisko, Farzam Gilani, Mark Wyse, Tommy Jung, Scott Davidson, Paul Gao, Chun Zhao, Zahra Azad, Sadullah Canakci, Bandhav Veluri, et al. "BlackParrot: An Agile Open Source RISC-V Multicore for Accelerator SoCs". In: *IEEE Micro* (2020).

[7] Yipeng Zhang, Bo Du, Lefei Zhang, and Jia Wu. "Parallel DNN Inference Framework Leveraging a Compact RISC-V ISA-based Multi-core System". In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2020, pp. 627–635.

[8] Dongkyu Lee, Hyeongyun Moon, Sejong Oh, and Daejin Park. "mIoT: Metamorphic IoT Platform for On-Demand Hardware Replacement in Large-Scaled IoT Applications". In: *Sensors 20*. 2020, p. 0.

[9] Mario Kovač, Philippe Notton, Daniel Hofman, and Josip Knezović. "How Europe is preparing its core solution for exascale machines and a global, sovereign, advanced computing platform". In: *Mathematical and Computational Applications 25*. 2020, p. 0.

[10] *IoT Signals report: IoT's promise will be unlocked by addressing skills shortage, complexity and security*. https://blogs.microsoft.com/blog/2019/07/30. Accessed: 2020-01-15.

[11] Colin Tankard. "Big data security". In: *Network security* 2012.7 (2012), pp. 5–8.

[12] Ramgopal Kashyap and Albert D Piersson. "Impact of big data on security". In: *Handbook of Research on Network Forensics and Analysis Techniques*. IGI Global, 2018, pp. 283–299.

[13] Sitalakshmi Venkatraman and Ramanathan Venkatraman. "Big data security challenges and strategies". In: *AIMS Mathematics* 4.3 (2019), pp. 860–879.

[14] Check Point Research. *Cyber Security Report 2020*. https://www.ntsc.org/assets/pdfs/cyber-security-report-2020.pdf. Online; accessed 8 December 2020.

[15] Ian Beer. *A very deep dive into iOS Exploit chains found in the wild*. https://googleprojectzero.blogspot.com/2019/08/a-very-deep-dive-into-ios-exploit.html. Online; accessed 16 December 2020.

[16] FireEye. *Unauthorized Access of FireEye Red Team Tools*. https://www.fireeye.com/blog/threat-research/2020/12/unauthorized-access-of-fireeye-red-team-tools.html. Online; accessed 9 December 2020.

[17] Casey Cane. *33 Alarming Cybercrime Statistics You Should Know in 2019*. https://securityboulevard.com/2019/11/33-alarming-cybercrime-statistics-you-should-know-in-2019. Online; accessed 9 December 2020.

[18] Ning Yu, Zachary Tuttle, Carl Jake Thurnau, and Emmanuel Mireku. "AI-Powered GUI Attack and Its Defensive Methods". In: *Proceedings of the 2020 ACM Southeast Conference*. 2020, pp. 79–86.

[19] Nektaria Kaloudi and Jingyue Li. "The ai-based cyber threat landscape: A survey". In: *ACM Computing Surveys (CSUR)* 53.1 (2020), pp. 1–34.

[20] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. "Meltdown". In: *arXiv preprint arXiv:1801.01207* (2018).

[21] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, et al. "Spectre attacks: Exploiting speculative execution". In: *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2019, pp. 1–19.

[22] David Williams-King, Hidenori Kobayashi, Kent Williams-King, Graham Patterson, Frank Spano, Yu Jian Wu, Junfeng Yang, and Vasileios P Kemerlis. "Egalito: Layout-Agnostic Binary Recompilation". In: *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. 2020, pp. 133–147.

[23] Angelo Garofalo, Manuele Rusci, Francesco Conti, Davide Rossi, and Luca Benini. "PULP-NN: A Computing Library for Quantized Neural Network inference at the edge on RISC-V Based Parallel Ultra Low Power Clusters". In: *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. IEEE. 2019, pp. 33–36.

[24] Chia-Hsiang Yang. "AI Acceleration with RISC-V for Edge Computing". In: *2020 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*. IEEE. 2020, pp. 1–1.

[25] Sebastian P Bayerl, Tommaso Frassetto, Patrick Jauernig, Korbinian Riedhammer, Ahmad-Reza Sadeghi, Thomas Schneider, Emmanuel Stapf, and Christian Weinert. "Offline model guard: Secure and private ML on mobile devices". In: *DATE 2020* (2020).

[26] Dominik Sisejkovic, Farhad Merchant, Lennart M Reimann, Harshit Srivastava, Ahmed Hallawa, and Rainer Leupers. "Challenging the Security of Logic Locking Schemes in the Era of Deep Learning: A Neuroevolutionary Approach". In: *arXiv preprint arXiv:2011.10389* (2020).

[27] Jaewon Lee, Hanning Chen, Jeffrey Young, and Hyesoon Kim. "RISC-V FPGA Platform Toward ROS-Based Robotics Application". In: *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE. 2020, pp. 370–370.

[28] Utsav Banerjee, Andrew Wright, Chiraag Juvekar, Madeleine Waller, Anantha P Chandrakasan, et al. "An energy-efficient reconfigurable dtls cryptographic engine for securing internet-of-things applications". In: *IEEE Journal of Solid-State Circuits* 54.8 (2019), pp. 2339–2352.

[29] Christian Palmiero, Giuseppe Di Guglielmo, Luciano Lavagno, and Luca P Carloni. "Design and implementation of a dynamic information flow tracking architecture to secure a RISC-V core for IoT applications". In: *2018 IEEE High Performance extreme Computing Conference (HPEC)*. IEEE. 2018, pp. 1–7.

[30] Hayate Takase, Ryotaro Kobayashi, Masahiko Kato, and Ren Ohmura. "A prototype implementation and evaluation of the malware detection mechanism for IoT devices using the processor information". In: (2020), p. 0.

[31] Dennis Agyemanh Nana Gookyi and Kwangki Ryoo. "Selecting a Synthesizable RISC-V Processor Core for Low-cost Hardware Devices". In: *Journal of Information Processing Systems* 15.6 (2019), pp. 1406–1421.

[32] Auer Lukas, Christian Skubich, and Matthias Hiller. "A Security Architecture for RISC-V based IoT Devices." In: *2019 Design Automation Test in Europe Conference Exhibition (DATE)*. 2019, pp. 1154–1159.

[33] Ckristian Duran, Megan Wachs, Albert Huntington, Javier Ardila, Jack Kang, Andres Amaya, Hector Gomez, Juan Romero, Laude Fernandez, Felipe Flechas, et al. "An Energy-Efficient RISC-V RV32IMAC Microcontroller for Periodical-Driven Sensing Applications". In: *2020 IEEE Custom Integrated Circuits Conference (CICC)*. IEEE. 2020, pp. 1–4.

[34] Vinay BY Kumar, Naina Gupta, Anupam Chattopadhyay, Michael Kasper, Christoph Krauß, and Ruben Niederhagen. "Post-quantum secure boot". In: *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2020, pp. 1582–1585.

[35] Vinay BY Kumar, Anupam Chattopadhyay, Jawad Haj-Yahya, and Avi Mendelson. "Itus: A secure risc-v system-on-chip". In: *2019 32nd IEEE International System-on-Chip Conference (SOCC)*. IEEE. 2019, pp. 418–423.

[36] Jawad Haj-Yahya, Ming Ming Wong, Vikramkumar Pudi, Shivam Bhasin, and Anupam Chattopadhyay. "Lightweight secure-boot architecture for risc-v system-on-chip". In: *20th International Symposium on Quality Electronic Design (ISQED)*. IEEE. 2019, pp. 216–223.

[37] A. S. Siddiqui, G. Shirley, S. Bendre, G. Bhagwat, J. Plusquellic, and F. Saqib. "Secure Design Flow of FPGA Based RISC-V Implementation". In: *2019 IEEE 4th International Verification and Security Workshop (IVSW)*. 2019, pp. 37–42.

[38] O. Arias, D. Sullivan, H. Shan, and Y. Jin. "SaeCAS: Secure Authenticated Execution Using CAM-Based Vector Storage". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39.11 (2020), pp. 4078–4089.

[39] Joris Jonkers Both, Patrick Spaans, Alexandru Geana, and Cees de Laat. "Analyzing and enhancing embedded software technologies on RISC-V64 using the Ghidra framework". In: (2020).

[40] Nate Graff Palmer Dabbelt. *SiFive's Trusted Execution Reference Platform*. 2018.

[41] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanović, and Dawn Song. "Keystone: An open framework for architecting trusted execution environments". In: *Proceedings of the Fifteenth European Conference on Computer Systems*. 2020, pp. 1–16.

[42] Cesare Garlati and Sandro Pinto. "A Clean Slate Approach to Linux Security RISC-V Enclaves". In: ().

[43] Arthur Azevedo De Amorim, Maxime Dénès, Nick Giannarakis, Catalin Hritcu, Benjamin C Pierce, Antal Spector-Zabusky, and Andrew Tolmach. "Micro-policies: Formally verified, tag-based security monitors". In: *2015 IEEE Symposium on Security and Privacy*. IEEE. 2015, pp. 813–830.

[44] Pascal Nasahl, Robert Schilling, Mario Werner, and Stefan Mangard. "HECTOR-V: A Heterogeneous CPU Architecture for a Secure RISC-V Execution Environment". In: *arXiv preprint arXiv:2009.05262* (2020).

[45] David Schrammel, Samuel Weiser, Stefan Steinegger, Martin Schwarzl, Michael Schwarz, Stefan Mangard, and Daniel Gruss. "Donky: Domain Keys–Efficient In-Process Isolation for RISC-V and x86". In: *29th {USENIX} Security Symposium ({USENIX} Security 20)*. 2020, pp. 1677–1694.

[46] Gui Andrade, Dayeol Lee, David Kohlbrenner, Krste Asanović, and Dawn Song. "Software-Based Off-Chip Memory Protection for RISC-V Trusted Execution Environments". In: ().

[47] Raad Bahmani, Ferdinand Brasser, Ghada Dessouky, Patrick Jauernig, Matthias Klimmek, Ahmad-Reza Sadeghi, and Emmanuel Stapf. "CURE: A Security Architecture with CUstomizable and Resilient Enclaves". In: *arXiv preprint arXiv:2010.15866* (2020).

[48] David Kohlbrenner, Shweta Shinde, Dayeol Lee, Krste Asanovic, and Dawn Song. "Building Open Trusted Execution Environments". In: *IEEE Security & Privacy* (2020).

[49] Moritz Schneider, Aritra Dhar, Ivan Puddu, Kari Kostiainen, and Srdjan Capkun. "PIE: A Dynamic TCB for Remote Systems with a Platform Isolation Environment". In: *arXiv preprint arXiv:2010.10416* (2020).

[50] Sandro Pinto and Nuno Santos. "Demystifying arm trustzone: A comprehensive survey". In: *ACM Computing Surveys (CSUR)* 51.6 (2019), pp. 1–36.

[51] David Cerdeira, Nuno Santos, Pedro Fonseca, and Sandro Pinto. "SoK: Understanding the prevailing security vulnerabilities in TrustZone-assisted TEE systems". In: *Proceedings of the IEEE Symposium on Security and Privacy (S&P), San Francisco, CA, USA*. 2020, pp. 18–20.

[52] A Waterman and K Asanovic. *The risc-v instruction set manual, volume ii: Privileged architecture, document version 20190608-priv-msu-ratified (2019)*.

[53] Kevin Cheang, Cameron Rasmussen, Dayeol Lee, David W Kohlbrenner, Krste Asanovic, and Sanjit A Seshia. "Verifying RISC-V Physical Memory Protection". In: ().

[54] H. Kim, J. Lee, D. Pratama, A. M. Awaludin, H. Kim, and D. Kwon. "RIMI: Instruction-level Memory Isolation for Embedded Systems on RISC-V". In: *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. 2020, pp. 1–9.

[55] Yier Jin. "Introduction to hardware security". In: *Electronics* 4 (2015), pp. 763–784.

[56] Wei Song, Alex Bradbury, and Robert Mullins. "Towards general purpose tagged memory". In: *Proceedings of the RISC-V Workshop. 2015*. 2015, p. 0.

[57] Mengyu Ma, Liwei Chen, and Gang Shi. "Dam: A Practical Scheme to Mitigate Data-Oriented Attacks with Tagged Memory Based on Hardware". In: *2019 26th Asia-Pacific Software Engineering Conference (APSEC)*. 2019, pp. 204–211.

[58] Zhenya Zang, Yao Liu, and Ray CC Cheung. "Reconfigurable risc-v secure processor and soc integration". In: *2019 IEEE International Conference on Industrial Technology (ICIT)*. IEEE. 2019, pp. 827–832.

[59] Arjun Menon, Subadra Murugan, Chester Rebeiro, Neel Gala, and Kamakoti Veezhinathan. "Shakti-T: A RISC-V processor with light weight security extensions". In: *Proceedings of the Hardware and Architectural Support for Security and Privacy*. 2017, pp. 1–8.

[60] Utsav Banerjee, Chiraag Juvekar, Andrew Wright, Anantha P Chandrakasan, et al. "An energy-efficient reconfigurable DTLS cryptographic engine for End-to-End security in iot applications". In: *2018 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE. 2018, pp. 42–44.

[61] Tong Liu, Gang Shi, Liwei Chen, Fei Zhang, Yaxuan Yang, and Jihu Zhang. "Tmdfi: Tagged memory assisted for fine-grained data-flow integrity towards embedded systems against software exploitation". In: *2018 17th IEEE International Conference On Trust Security And Privacy Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. 2018, pp. 545–550.

[62] Jiyong Yu, Lucas Hsiung, Mohamad El Hajj, and Christopher W Fletcher. "Creating Foundations for Secure Microarchitectures With Data-Oblivious ISA Extensions." In: *IEEE Micro* 40.3 (2020), pp. 99–107.

[63] Lejla Batina, Patrick Jauernig, Nele Mentens, Ahmad-Reza Sadeghi, and Emmanuel Stapf. "In Hardware We Trust: Gains and Pains of Hardware-assisted Security". In: *2019 56th ACM/IEEE Design Automation Conference (DAC)*. IEEE. 2019, pp. 1–4.

[64] Klaus v Gleissenthall, Rami Gökhan Kıcı, Deian Stefan, and Ranjit Jhala. "{IODINE}: Verifying Constant-Time Execution of Hardware". In: *28th {USENIX} Security Symposium ({USENIX} Security 19)*. 2019, pp. 1411–1428.

[65] Asmit De, Aditya Basu, Swaroop Ghosh, and Trent Jaeger. "Hardware Assisted Buffer Protection Mechanisms for Embedded RISC-V". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2020).

[66] Nils Wistoff, Moritz Schneider, Frank K Gürkaynak, Luca Benini, and Gernot Heiser. "Prevention of Microarchitectural Covert Channels on an Open-Source 64-bit RISC-V Core". In: *arXiv preprint arXiv:2005.02193* (2020).

[67] Gernot Heiser, Toby Murray, and Gerwin Klein. "Towards Provable Timing-Channel Prevention". In: *ACM SIGOPS Operating Systems Review* 54 (2020), pp. 1–7.

[68] Anish Athalye, Adam Belay, M Frans Kaashoek, Robert Morris, and Nickolai Zeldovich. "Notary: a device for secure transaction approval". In: *Proceedings of the 27th ACM Symposium on Operating Systems Principles*. 2019, pp. 97–113.

[69] Maja Malenko and Marcel Baunach. "Device Driver and System Call Isolation in Embedded Devices". In: *2019 22nd Euromicro Conference on Digital System Design (DSD)*. IEEE. 2019, pp. 283–290.

[70] Olivier Savry, Mustapha El-Majihi, and Thomas Hiscock. "Confidaent: Control FLow protection with Instruction and Data Authenticated Encryption". In: *2020 23rd Euromicro Conference on Digital System Design (DSD)*. 2020, pp. 246–253.

[71] Samuel Weiser, Mario Werner, Ferdinand Brasser, Maja Malenko, Stefan Mangard, and Ahmad-Reza Sadeghi. "TIMBER-V: Tag-Isolated Memory Bringing Fine-grained Enclaves to RISC-V." In: *NDSS*. 2019.

[72] Kyndylan Nienhuis, Alexandre Joannou, Thomas Bauereiss, Anthony Fox, Michael Roe, Brian Campbell, Matthew Naylor, Robert M Norton, Simon W Moore, Peter G Neumann, et al. "Rigorous engineering for hardware security: Formal modelling and proof in the CHERI design and implementation process". In: *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2020, pp. 1003–1020.

[73] Tuo Li, Bradley Hopkins, and Sri Parameswaran. "SIMF: Single-Instruction Multiple-Flush Mechanism for Processor Temporal Isolation". In: *arXiv preprint arXiv:2011.10249*. 2020, p. 0.

[74] Mathieu Escouteloup, Jacques Fournier, Jean-Louis Lanet, and Ronan Lashermes. "Recommendations for a radically secure ISA". In: *Workshop on Computer Architecture Research with RISC-V*. 2020.

[75] Jiyong Yu, Lucas Hsiung, Mohamad El'Hajj, and Christopher W Fletcher. "Data oblivious ISA extensions for side channel-resistant and high performance computing". In: *The Network and Distributed System Security Symposium (NDSS)*. 2019.

[76] Robert NM Watson, Peter G Neumann, Jonathan Woodruff, Michael Roe, Jonathan Anderson, David Chisnall, Brooks Davis, Alexandre Joannou, Ben Laurie, Simon W Moore, et al. *Capability hardware enhanced risc instructions: Cheri instruction-set architecture (version 8)*. Tech. rep. University of Cambridge, Computer Laboratory, 2020.

[77] Markku-Juhani O Saarinen. "SNEIK on Microcontrollers: AVR, ARMv7-M, and RISC-V with Custom Instructions." In: *IACR Cryptol. ePrint Arch.* 2019 (2019), p. 936.

[78] Etienne Tehrani, Tarik Graba, Abdelmalek Si Merabet, Sylvain Guilley, and Jean-Luc Danger. "Classification of Lightweight Block Ciphers for Specific Processor Accelerated Implementations". In: *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. IEEE. 2019, pp. 747–750.

[79] E. Tehrani, T. Graba, A. S. Merabet, and J. L. Danger. "RISC-V Extension for Lightweight Cryptography". In: *2020 23rd Euromicro Conference on Digital System Design (DSD)*. 2020, pp. 222–228.

[80] Bastian Koppelmann, Peer Adelt, Wolfgang Mueller, and Christoph Scheytt. "RISC-V Extensions for Bit Manipulation Instructions". In: *2019 29th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*. IEEE. 2019, pp. 41–48.

[81] Erdem Alkim, Hülya Evkan, Norman Lahr, Ruben Niederhagen, and Richard Petri. "ISA Extensions for Finite Field Arithmetic". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2020), pp. 219–242.

[82] Ben Marshall, G Richard Newell, Dan Page, Markku-Juhani O Saarinen, and Claire Wolf. *The design of scalar AES Instruction Set Extensions for RISC-V*. Tech. rep. Cryptology ePrint Archive, Report 2020/930. https://eprint. iacr. org/2020/930, 2020.

[83] Alexander Zeh, Andy Glew, Barry Spinney, Ben Marshall, Daniel Page, Derek Atkins, Ken Dockser, Markku-Juhani O Saarinen, Nathan Menhorn, Richard Newell, et al. "RISC-V Cryptographic Extension Proposals Volume I: Scalar & Entropy Source Instructions". In: ().

[84] Kotaro Matsuoka, Ryotaro Banno, Naoki Matsumoto, Takashi Sato, and Song Bian. "Virtual Secure Platform: A Five-Stage Pipeline Processor over TFHE". In: *arXiv preprint arXiv:2010.09410*. 2020, p. 0.

[85] Kartik Nayak, Christopher W Fletcher, Ling Ren, Nishanth Chandran, Satya V Lokam, Elaine Shi, and Vipul Goyal. "HOP: Hardware makes Obfuscation Practical". In: *NDSS. 2017*. 2017, p. 0.

[86] Jinli Rao, Tianyong Ao, Shu Xu, Kui Dai, and Xuecheng Zou. "Design Exploration of SHA-3 ASIP for IoT on a 32-bit RISC-V Processor". In: *IEICE TRANSACTIONS on Information and Systems* 101.11 (2018), pp. 2698–2705.

[87] Markku-Juhani O Saarinen, G Richard Newell, and Ben Marshall. "Building a Modern TRNG: An Entropy Source Interface for RISC-V". In: *Proceedings of the 4th ACM Workshop on Attacks and Solutions in Hardware Security*. 2020, pp. 93–102.

[88] Kevin Kiningham, Philip Levis, Mark Anderson, Dan Boneh, Mark Horowitz, and Maurice Shih. "Falcon—A Flexible Architecture For Accelerating Cryptography". In: *2019 IEEE 16th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*. 2019, pp. 136–144.

[89] Venkat SK Balagurusamy, Cyril Cabral, Srikumar Coomaraswamy, Emmanuel Delamarche, Donna N Dillenberger, Gero Dittmann, and Daniel Friedman et al. "Crypto anchors". In: (2019), p. 0.

[90] Lukas Jager and Richard Petri. "DICE harder: a hardware implementation of the device identifier composition engine". In: *Proceedings of the 15th International Conference on Availability*. 2020, pp. 1–8.

[91] Antoine Loiseau and Jacques JA Fournier. "Binary Edwards Curves for Intrinsically Secure ECC Implementations for the IoT." In: *ICETE (2)*. 2018, pp. 625–631.

[92] Lars Jellema. *optimizing Ascon on RISC-V*. 2019.

[93] Ömer Faruk Irmak and Arda Yurdakul. "An Embedded RISC-V Core with Fast Modular Multiplication". In: *arXiv e-prints* (2020), arXiv–2009.

[94] Stefan Steinegger and Robert Primas. "A Fast and Compact RISC-V Accelerator for Ascon and Friends". In: *CARDIS 2020: 19th Smart Card Research and Advanced Application Conference*. 2020.

[95] Rui Zhang and Cynthia Sturton. "Transys: Leveraging Common Security Properties Across Hardware Designs". In: *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2020, pp. 1713–1727.

[96] Debapriya Basu Roy, Tim Fritzmann, and Georg Sigl. "Efficient Hardware/Software Co-Design for Post-Quantum Crypto Algorithm SIKE on ARM and RISC-V based Microcontrollers". In: *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE. 2020, pp. 1–9.

[97] Campos Fabio, Lars Jellema, Mauk Lemmen, Lars Müller, Daan Sprenkels, and Benoit Viguier. "Assembly or Optimized C for Lightweight Cryptography on RISC-V?" In: *Cryptology ePrint Archive*. 2020, p. 0.

[98] Ko Stoffelen. "Efficient cryptography on the RISC-V architecture". In: *International Conference on Cryptology and Information Security in Latin America*. Springer. 2019, pp. 323–340.

[99] Shuwen Deng, Wenjie Xiong, and Jakub Szefer. "Secure tlbs". In: *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*. IEEE. 2019, pp. 346–359.

[100] Abraham Gonzalez, Ben Korpan, Jerry Zhao, Ed Younis, and Krste Asanović. "Replicating and Mitigating Spectre Attacks on an Open Source RISC-V Microarchitecture". In: *Third Workshop on Computer Architecture Research with RISC-V (CARRV 2019), Phoenix, AZ, USA*. 2019.

[101] Otávio Oliveira Napoli, Vanderson Martins do Rosario, Diego de Freitas Aranha, and Edson Borin. "Evaluation of timing side-channel leakage on a multiple-target dynamic binary translator". In: *2018 Symposium on High Performance Computing Systems (WSCAD)*. 2018, pp. 198–204.

[102] Cezar Reinbrecht, Abdullah Aljuffri, Said Hamdioui, Mottaqiallah Taouil, Bruno Forlin, and Johanna Sepulveda. "Guard-NoC: A Protection Against Side-Channel Attacks for MPSoCs". In: *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. 2020, pp. 536–541.

[103] Elke De Mulder, Samatha Gummalla, and Michael Hutter. "Protecting RISC-V against side-channel attacks". In: *2019 56th ACM/IEEE Design Automation Conference (DAC)*. IEEE. 2019, pp. 1–4.

[104] Sheng Zhang, Adrian Tang, Zhewei Jiang, Simha Sethumadhavan, and Mingoo Seok. "Blacklist core: machine-learning based dynamic operating-performance-point blacklisting for mitigating power-management security attacks". In: *Proceedings of the International Symposium on Low Power Electronics and Design*. 2018, pp. 1–6.

[105] Ba-Anh Dao, Anh-Tien Le, Trong-Thuc Hoang, Akira Tsukamoto, Kuniyasu Suzaki, and Cong-Kha Pham. "Dynamic Frequency Scaling as a countermeasure against simple power analysis attack in RISC-V processors". In: 2020, p. 0.

[106] Marno van der Maas and Simon W Moore. "Protecting Enclaves from Intra-Core Side-Channel Attacks through Physical Isolation". In: *Proceedings of the 2nd Workshop on Cyber-Security Arms Race*. 2020, pp. 1–12.

[107] Laurent Sauvage, Sofiane Takarabt, and Youssef Souissi. "Secure silicon: Towards virtual prototyping". In: *2017 International Symposium on Electromagnetic Compatibility-EMC EUROPE*. IEEE. 2017, pp. 1–5.

[108] Pasquale Davide Schiavone, Francesco Conti, Davide Rossi, Michael Gautschi, Antonio Pullini, Eric Flamand, and Luca Benini. "Slow and steady wins the race? A comparison of ultra-low-power RISC-V cores for Internet-of-Things applications". In: *2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*. IEEE. 2017, pp. 1–8.

[109] Western Digital. *RISC-V and Open Source Hardware Address New Compute Requirements*. https://documents.westerndigital.com/content/dam/doc-library/en_us/assets/public/western-digital/collateral/tech-brief/tech-brief-western-digital-risc-v.pdf. Online; accessed 7 May 2021.

[110] Eric Flamand, Davide Rossi, Francesco Conti, Igor Loi, Antonio Pullini, Florent Rotenberg, and Luca Benini. "GAP-8: A RISC-V SoC for AI at the Edge of the IoT". In: *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE. 2018, pp. 1–4.

[111] Marcia Sahaya Louis, Zahra Azad, Leila Delshadtehrani, Suyog Gupta, Pete Warden, Vijay Janapa Reddi, and Ajay Joshi. "Towards deep learning using tensorflow lite on risc-v". In: *Third Workshop on Computer Architecture Research with RISC-V (CARRV)*. Vol. 1. 2019, p. 6.

[112] Angelo Garofalo, Manuele Rusci, Francesco Conti, Davide Rossi, and Luca Benini. "PULP-NN: accelerating quantized neural networks on parallel ultra-low-power RISC-V processors". In: *Philosophical Transactions of the Royal Society A* 378 (2020), p. 20190155.

[113] Chen Chen, Xiaoyan Xiang, Chang Liu, Yunhai Shang, Ren Guo, Dongqi Liu, Yimin Lu, Ziyi Hao, Jiahui Luo, Zhijian Chen, et al. "Xuantie-910: A commercial multi-core 12-stage pipeline out-of-order 64-bit high performance risc-v processor with vector extension: Industrial product". In: *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE. 2020, pp. 52–64.

[114] Matheus Cavalcante, Fabian Schuiki, Florian Zaruba, Michael Schaffner, and Luca Benini. "Ara: A 1-ghz+ scalable and energy-efficient risc-v vector processor with multiprecision floating-point support in 22-nm fd-soi". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 28.2 (2019), pp. 530–543.

[115] SiFive. *SiFive Performance P550 Core Sets New Standard as Highest Performance RISC-V Processor IP*. https://www.sifive.com/press/sifive-performance-p550-core-sets-new-standard-as-highest. Online; accessed 24 June 2021.

[116] The Regents of the University of California (Regents). *RISC-V GNU Compiler Toolchain*. https://github.com/riscv/riscv-gnu-toolchain. Online; accessed 24 June 2021.

[117] SiFive. *Freedom Studio Version 2019.03*. https://www.sifive.com/blog/freedom-studio-version-2019.03. Online; accessed 24 June 2021.

[118] *Major breach found in biometrics system used by banks, UK police and defence firms*. 2019. eprint: https://www.theguardian.com/technology/2019/aug/14/major-breach-found-in-biometrics-system-used-by-banks-uk-police-and-defence-firms.
[119] Craig Gentry. "Fully homomorphic encryption using ideal lattices". In: *Proceedings of the forty-first annual ACM symposium on Theory of computing*. 2009, pp. 169–178.
[120] Oded Goldreich. "Secure multi-party computation". In: *Manuscript. Preliminary version* 78 (1998).
[121] Ying Bai. "ARM® Memory Protection Unit (MPU)". In: (2016).
[122] Victor Costan and Srinivas Devadas. "Intel sgx explained." In: *IACR Cryptol. ePrint Arch.* 2016.86 (2016), pp. 1–118.
[123] Karim Eldefrawy, Gene Tsudik, Aurélien Francillon, and Daniele Perito. "SMART: Secure and Minimal Architecture for (Establishing Dynamic) Root of Trust." In: *Ndss*. Vol. 12. 2012, pp. 1–15.
[124] Alan Ehret, Eliakin Del Rosario, Karen Gettings, and Michel A Kinsy. "A Hardware Root-of-Trust Design for Low-Power SoC Edge Devices". In: *2020 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE. 2020, pp. 1–6.
[125] Marko Wolf and Timo Gendrullis. "Design, implementation, and evaluation of a vehicular hardware security module". In: *International Conference on Information Security and Cryptology*. Springer. 2011, pp. 302–318.
[126] Norm Robson, John Safran, Chandrasekharan Kothandaraman, Alberto Cestero, Xiang Chen, Raj Rajeevakumar, Alan Leslie, Dan Moy, Toshiaki Kirihata, and Subramanian Iyer. "Electrically programmable fuse (efuse): From memory redundancy to autonomic chips". In: *2007 IEEE Custom Integrated Circuits Conference*. IEEE. 2007, pp. 799–804.
[127] ARM. *Memory Protection Unit (MPU) Version 1.0*. https://developer.arm.com/documentation/100699/0100. Online; accessed 14 December 2020.
[128] Kevin Cheang, Cameron Rasmussen, Dayeol Lee, David W Kohlbrenner, Krste Asanovic, and Sanjit A Seshia. "Verifying RISC-V Physical Memory Protection". In: (2020).
[129] Ilia Lebedev, Kyle Hogan, and Srinivas Devadas. "Secure boot and remote attestation in the sanctum processor". In: *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*. IEEE. 2018, pp. 46–60.
[130] James Nechvatal, Elaine Barker, Lawrence Bassham, William Burr, Morris Dworkin, James Foti, and Edward Roback. "Report on the development of the Advanced Encryption Standard (AES)". In: *Journal of Research of the National Institute of Standards and Technology* 106.3 (2001), p. 511.
[131] Whitfield Diffie. "The first ten years of public-key cryptography". In: *Proceedings of the IEEE* 76.5 (1988), pp. 560–577.
[132] Nan Li. "Research on Diffie-Hellman key exchange protocol". In: *2010 2nd International Conference on Computer Engineering and Technology*. Vol. 4. IEEE. 2010, pp. V4–634.
[133] Jakob Jonsson and Burt Kaliski. *Public-key cryptography standards (PKCS)# 1: RSA cryptography specifications version 2.1*. Tech. rep. RFC 3447, February, 2003.
[134] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. *HMAC: Keyed-hashing for message authentication*. 1997.
[135] Don Johnson, Alfred Menezes, and Scott Vanstone. "The elliptic curve digital signature algorithm (ECDSA)". In: *International journal of information security* 1.1 (2001), pp. 36–63.
[136] Viktor Fischer and Miloš Drutarovskỳ. "True random number generator embedded in reconfigurable hardware". In: *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer. 2002, pp. 415–430.
[137] Elaine Barker and John Kelsey. "NIST Special Publication 800-90A Recommendation for Random Number Generation Using Deterministic Random Bit Generators". In: (2012).
[138] Yuan Liu, Jed Briones, Ruolin Zhou, and Neeraj Magotra. "Study of secure boot with a FPGA-based IoT device". In: *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*. IEEE. 2017, pp. 1053–1056.
[139] ARM. *Learn the architecture: AArch64 Exception model*. 2020. eprint: https://developer.arm.com/documentation/102412/0100/Privilege-and-Exception-levels.
[140] ARM. *Arm Architecture Reference Manual Armv8, for Armv8-A architecture profile*. https://developer.arm.com/documentation/ddi0487/ga. Online; accessed 6 May 2021.
[141] Mauro Olivieri, Abdallah Cheikh, Gianmarco Cerutti, Antonio Mastrandrea, and Francesco Menichelli. "Investigation on the optimal pipeline organization in RISC-V multi-threaded soft processor cores". In: *2017 New Generation of CAS (NGCAS)*. IEEE. 2017, pp. 45–48.
[142] Christopher Celio, Pi-Feng Chiu, Borivoje Nikolic, David A Patterson, and Krste Asanovic. "BOOMv2: an open-source out-of-order RISC-V core". In: *First Workshop on Computer Architecture Research with RISC-V (CARRV)*. 2017.
[143] Krste Asanovic, Rimas Avizienis, Jonathan Bachrach, Scott Beamer, David Biancolin, Christopher Celio, Henry Cook, Daniel Dabbelt, John Hauser, Adam Izraelevitz, et al. "The rocket chip generator". In: *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17* (2016).

[144] M. R. Fadiheh, D. Stoffel, C. Barrett, S. Mitra, and W. Kunz. "Processor Hardware Security Vulnerabilities and their Detection by Unique Program Execution Checking". In: *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*. 2019, pp. 994–999.

[145] Mahmoud A Elmohr, Haohao Liao, and Catherine H Gebotys. "Em fault injection on arm and risc-v". In: *2020 21st International Symposium on Quality Electronic Design (ISQED)*. IEEE. 2020, pp. 206–212.

[146] Seetal Potluri, Aydin Aysu, and Akash Kumar. "SeqL: Secure Scan-Locking for IP Protection". In: *arXiv preprint arXiv:2005.13032* (2020).

[147] Martin Maas, Eric Love, Emil Stefanov, Mohit Tiwari, Elaine Shi, Krste Asanovic, John Kubiatowicz, and Dawn Song. "Phantom: Practical oblivious computation in a secure processor". In: *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 2013, pp. 311–324.

[148] Mario Werner, Robert Schilling, Thomas Unterluggauer, and Stefan Mangard. "Protecting risc-v processors against physical attacks". In: *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2019, pp. 1136–1141.

[149] Timothy Linscott, Pete Ehrett, Valeria Bertacco, and Todd Austin. "SWAN: Mitigating hardware trojans with design ambiguity". In: *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE. 2018, pp. 1–7.

[150] Junko Takahashi, Keiichi Okabe, Hiroki Itoh, Xuan-Thuy Ngo, Sylvain Guilley, Ritu-Ranjan Shrivastwa, Mushir Ahmed, and Patrick Lejoly. "Machine Learning Based Hardware Trojan Detection Using Electromagnetic Emanation". In: *International Conference on Information and Communications Security*. Springer. 2020, pp. 3–19.

[151] A. Bolat, L. Cassano, P. Reviriego, O. Ergin, and M. Ottavi. "A Microprocessor Protection Architecture against Hardware Trojans in Memories". In: *2020 15th Design Technology of Integrated Systems in Nanoscale Era (DTIS)*. 2020, pp. 1–6.

[152] Ghada Dessouky, David Gens, Patrick Haney, Garrett Persyn, Arun Kanuparthi, Hareesh Khattri, Jason M Fung, Ahmad-Reza Sadeghi, and Jeyavijayan Rajendran. "Hardfails: Insights into software-exploitable hardware bugs". In: *28th USENIX Security Symposium (USENIX Security 19)*. 2019, pp. 213–230.

[153] Mohammad Rahmani Fadiheh, Dominik Stoffel, Clark Barrett, Subhasish Mitra, and Wolfgang Kunz. "Processor hardware security vulnerabilities and their detection by unique program execution checking". In: *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2019, pp. 994–999.

[154] Ahmad-Reza Sadeghi, Jeyavijayan Rajendran, and Rahul Kande. "Organizing The World's Largest Hardware Security Competition: Challenges, Opportunities, and Lessons Learned". In: *Proceedings of the 2021 on Great Lakes Symposium on VLSI*. 2021, pp. 95–100.

[155] Kan Xiao, Domenic Forte, Yier Jin, Ramesh Karri, Swarup Bhunia, and Mohammad Tehranipoor. "Hardware trojans: Lessons learned after one decade of research". In: *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 22.1 (2016), pp. 1–23.

[156] Alexander Hepp and Georg Sigl. "Tapeout of a RISC-V crypto chip with hardware trojans: a case-study on trojan design and pre-silicon detectability". In: *Proceedings of the 18th ACM International Conference on Computing Frontiers*. 2021, pp. 213–220.

[157] Yusuf Kulah, Berkay Dincer, Cemal Yilmaz, and Erkay Savas. "SpyDetector: An approach for detecting side-channel attacks at runtime". In: *International Journal of Information Security* 18.4 (2019), pp. 393–422.

[158] Jiaji He, Yiqiang Zhao, Xiaolong Guo, and Yier Jin. "Hardware trojan detection through chip-free electromagnetic side-channel statistical analysis". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 25.10 (2017), pp. 2939–2948.

[159] Tamzidul Hoque, Xinmu Wang, Abhishek Basak, Robert Karam, and Swarup Bhunia. "Hardware trojan attacks in embedded memory". In: *2018 IEEE 36th VLSI Test Symposium (VTS)*. IEEE. 2018, pp. 1–6.

[160] Krste Asanović, Rimas Avizienis, Jonathan Bachrach, Scott Beamer, David Biancolin, Christopher Celio, Henry Cook, Daniel Dabbelt, John Hauser, Adam Izraelevitz, Sagar Karandikar, Ben Keller, Donggyu Kim, John Koenig, Yunsup Lee, Eric Love, Martin Maas, Albert Magyar, Howard Mao, Miquel Moreto, Albert Ou, David A. Patterson, Brian Richards, Colin Schmidt, Stephen Twigg, Huy Vo, and Andrew Waterman. *The Rocket Chip Generator*. Tech. rep. EECS Department, University of California, Berkeley, 2016.

[161] Arnab Kumar Biswas, Dipak Ghosal, and Shishir Nagaraja. "A survey of timing channels and countermeasures". In: *ACM Computing Surveys (CSUR)* 50.1 (2017), pp. 1–39.

[162] Huanyu Wang, Domenic Forte, Mark M Tehranipoor, and Qihang Shi. "Probing attacks on integrated circuits: Challenges and research opportunities". In: *IEEE Design & Test* 34.5 (2017), pp. 63–71.

[163] Mark Gallagher, Lauren Biernacki, Shibo Chen, Zelalem Birhanu Aweke, Salessawi Ferede Yitbarek, Misiker Tadesse Aga, Austin Harris, Zhixing Xu, Baris Kasikci, Valeria Bertacco, et al. "Morpheus: a vulnerability-tolerant secure architecture based on ensembles of moving target defenses with churn". In: *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 2019, pp. 469–484.

[164] Luigi Coppolino, Salvatore D'Antonio, Giovanni Mazzeo, and Luigi Romano. "A comprehensive survey of hardware-assisted security: From the edge to the cloud". In: *Internet of Things* 6 (2019), p. 100055.

[165] Wen Wang, Bernhard Jungk, Julian Wälde, Shuwen Deng, Naina Gupta, Jakub Szefer, and Ruben Niederhagen. "XMSS and embedded systems". In: *International Conference on Selected Areas in Cryptography*. Springer. 2019, pp. 523–550.

[166] Nickolai Zeldovich, Hari Kannan, Michael Dalton, and Christos Kozyrakis. "Hardware Enforcement of Application Security Policies Using Tagged Memory." In: *OSDI*. Vol. 8. 2008, pp. 225–240.

[167] Alexandre Joannou, Jonathan Woodruff, Robert Kovacsics, Simon W Moore, Alex Bradbury, Hongyan Xia, Robert NM Watson, David Chisnall, Michael Roe, Brooks Davis, et al. "Efficient tagged memory". In: *2017 IEEE International Conference on Computer Design (ICCD)*. IEEE. 2017, pp. 641–648.

[168] Robert NM Watson, Alexander Richardson, Brooks Davis, John Baldwin, David Chisnall, Jessica Clarke, Nathaniel Filardo, Simon W Moore, Edward Napierala, Peter Sewell, et al. *CHERI C/C++ Programming Guide*. Tech. rep. University of Cambridge, Computer Laboratory, 2020.

[169] Long Cheng, Hans Liljestrand, Md Salman Ahmed, Thomas Nyman, Trent Jaeger, N Asokan, and Danfeng Yao. "Exploitation techniques and defenses for data-oriented attacks". In: *2019 IEEE Cybersecurity Development (SecDev)*. IEEE. 2019, pp. 114–128.

[170] David Gens, Simon Schmitt, Lucas Davi, and Ahmad-Reza Sadeghi. "K-Miner: Uncovering Memory Corruption in Linux." In: *NDSS*. 2018.

[171] Claire Wolf. *Risc-v bitmanip (bit manipulation) extension, document version 0.94-draft*. 2021.

[172] Cryptographic Hashing. "SNEIKEN and SNEIKHA". In: (2019).

[173] Martín Abadi, Mihai Budiu, Ulfar Erlingsson, and Jay Ligatti. "Control-flow integrity principles, implementations, and applications". In: *ACM Transactions on Information and System Security (TISSEC)* 13.1 (2009), pp. 1–40.

[174] Pratyusa K Manadhata and Jeannette M Wing. "An attack surface metric". In: *IEEE Transactions on Software Engineering* 37.3 (2010), pp. 371–386.

[175] Bernhard Kauer. "OSLO: Improving the Security of Trusted Computing." In: *USENIX Security Symposium*. Vol. 24. 2007, p. 173.

[176] Muhammad Arif, Guojun Wang, and Valentina Emilia Balas. "Secure VANETs: trusted communication scheme between vehicles and infrastructure based on fog computing". In: *Stud. Inform. Control* 27.2 (2018), pp. 235–246.

[177] Gagangeet Singh Aujla, Rajat Chaudhary, Neeraj Kumar, Ashok Kumar Das, and Joel JPC Rodrigues. "SecSVA: secure storage, verification, and auditing of big data in the cloud environment". In: *IEEE Communications Magazine* 56.1 (2018), pp. 78–85.

[178] Taizo Shirai, Kyoji Shibutani, Toru Akishita, Shiho Moriai, and Tetsu Iwata. "The 128-bit blockcipher CLEFIA". In: *International workshop on fast software encryption*. Springer. 2007, pp. 181–195.

[179] Carlos Andres Lara-Nino, Miguel Morales-Sandoval, and Arturo Diaz-Perez. "Novel FPGA-based low-cost hardware architecture for the PRESENT block cipher". In: *2016 Euromicro Conference on Digital System Design (DSD)*. IEEE. 2016, pp. 646–650.

[180] Deukjo Hong, Jung-Keun Lee, Dong-Chan Kim, Daesung Kwon, Kwon Ho Ryu, and Dong-Geon Lee. "LEA: A 128-bit block cipher for fast encryption on common processors". In: *International Workshop on Information Security Applications*. Springer. 2013, pp. 3–27.

[181] Lily Chen, Stephen Jordan, Yi-Kai Liu, Dustin Moody, Rene Peralta, Ray Perlner, and Daniel Smith-Tone. *Report on post-quantum cryptography*. Vol. 12. US Department of Commerce, National Institute of Standards and Technology, 2016.

[182] Dustin Moody, Gorjan Alagic, Daniel C Apon, David A Cooper, Quynh H Dang, John M Kelsey, Yi-Kai Liu, Carl A Miller, Rene C Peralta, Ray A Perlner, et al. "Status Report on the Second Round of the NIST Post-Quantum Cryptography Standardization Process". In: (2020).

[183] Jürgen Teich. "Hardware/software codesign: The past, the present, and predicting the future". In: *Proceedings of the IEEE* 100.Special Centennial Issue (2012), pp. 1411–1430.

[184] Mark Zhao and G Edward Suh. "FPGA-based remote power side-channel attacks". In: *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2018, pp. 229–244.

[185] Dakshi Agrawal, Bruce Archambeault, Josyula R Rao, and Pankaj Rohatgi. "The EM side—channel (s)". In: *International workshop on cryptographic hardware and embedded systems*. Springer. 2002, pp. 29–45.

[186] Wenlei Bao, Changwan Hong, Sudheer Chunduri, Sriram Krishnamoorthy, Louis-Noël Pouchet, Fabrice Rastello, and P Sadayappan. "Static and dynamic frequency scaling on multicore CPUs". In: *ACM Transactions on Architecture and Code Optimization (TACO)* 13.4 (2016), pp. 1–26.

[187] Adrian Tang, Simha Sethumadhavan, and Salvatore Stolfo. "{CLKSCREW}: exposing the perils of security-oblivious energy management". In: *26th {USENIX} Security Symposium ({USENIX} Security 17)*. 2017, pp. 1057–1074.

[188] Ben Gras, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. "Translation leak-aside buffer: Defeating cache side-channel protections with {TLB} attacks". In: *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 2018, pp. 955–972.

[189] Marcelo Ruaro, Luciano L Caimi, Vinicius Fochi, and Fernando G Moraes. "Memphis: a framework for heterogeneous many-core socs generation and validation". In: *Design Automation for Embedded Systems* 23.3 (2019), pp. 103–122.

[190] Arnab Kumar Biswas. "Cryptographic software IP protection without compromising performance or timing side-channel leakage". In: *ACM Transactions on Architecture and Code Optimization (TACO)* 18.2 (2021), pp. 1–20.

[191] Andreas Traber, Florian Zaruba, Sven Stucki, Antonio Pullini, Germain Haugou, Eric Flamand, Frank K Gurkaynak, and Luca Benini. "PULPino: A small single-core RISC-V SoC". In: *3rd RISCV Workshop*. 2016.

[192] Shoei Nashimoto, Daisuke Suzuki, Rei Ueno, and Naofumi Homma. "Bypassing Isolated Execution on RISC-V with Fault Injection". In: (2020).