A Security Architecture for RISC-V based IoT Devices

Lukas Auer*, Christian Skubich†, and Matthias Hiller*
*Fraunhofer Institute for Applied and Integrated Security AISEC, Garching near Munich, Germany
†Fraunhofer Institute for Integrated Circuits IIS, Division Engineering of Adaptive Systems EAS, Dresden, Germany
{forename.surname}@{aisec.fraunhofer.de,eas.iis.fraunhofer.de}

***Abstract*—New IoT applications are demanding for more and more performance in embedded devices while their deployment and operation poses strict power constraints. We present the security concept for a customizable Internet of Things (IoT) platform based on the RISC-V ISA and developed by several Fraunhofer Institutes. It integrates a range of peripherals with a scalable computing subsystem as a three dimensional System-in-Package (3D-SiP).**

The security features aim for a medium security level and target the requirements of the IoT market. Our security architecture extends given implementations to enable secure deployment, operation, and update. Core security features are secure boot, an authenticated watchdog timer, and key management.

The Universal Sensor Platform (USeP) SoC is developed for GLOBALFOUNDRIES' 22FDX technology and aims to provide a platform for Small and Medium-sized Enterprises (SMEs) that typically do not have access to advanced microelectronics and integration know-how, and are therefore limited to Commercial Off-The-Shelf (COTS) products.

***Index Terms*—RISC-V, device security, secure boot, watchdog timer, IoT.**

I. Introduction

IoT devices enable new services, business models, and consumer experiences in many different areas of life so that their adoption is rapidly increasing [1]. They sense and interact with their environment which generates and consumes data. The increasing amounts of data that are generated on the device require the computational efforts to be moved from distant clouds to the edge of the network due to power and bandwidth limitations.

Many IoT devices are designed to be self-contained, autonomous systems, which do not rely on human interaction. This makes it possible to deploy a large amount of devices, many in remote out-of-reach locations. To support this use-case, the devices' life-span must be sufficiently long to avoid short maintenance intervals. A primary requirement is to maintain the devices' security over their entire life-span. The draft for the NIST Internal Report 8228 [2] lists the following three high-level mitigation goals:

- Device security must be ensured through-out its life-time to prevent devices from being used as part of attacks. This requires to identify relevant vulnerabilities in the system, monitor IoT devices as part of asset tracking and to detect possible security incidents, and access management.
- Data on devices must be protected, to prevent exposure of sensitive information. Monitoring and analysis is required to detect security incidents.
- To protect the privacy of individuals, personally identifiable information must be protected in general. If possible, the information should be disassociated from individuals. Users must be able to understand if and how information is processed. The device activity must be monitored to detect possible privacy breaches.

These goals can only be met by developing the security concept and mechanisms in the early design stages.

To achieve a competitive level of performance, integration, power consumption, and security extensive microelectronics and integration know-how is required. This leads to high development costs, which are hard to bear for Small and Medium-sized Enterprises (SMEs).

A major challenge regarding IoT hardware is the system integration. Customers expect IoT devices to be very compact, which requires integration technologies beyond Printed Circuit Board (PCB) level integration. While there are solutions available that integrate core components such as a processor, radio interface, and memory in a System in Package (SiP), the number of sensors is typically limited, and the selection of components may be restricted and unable to fit the target application. Customizing a SiP is therefore, typically, only feasible for high volume applications of large manufacturers.

For example, zGlue is a project that strives to change this and provide a heterogeneous integration platform. Using system design automation and programmable interconnects, it aims to provide a shorter time-to-market at lower costs compared with traditional SiP technology [3].

Our goal is to take integration one step further and create a three dimensional System-in-Package (3D-SiP) for the IoT market. It integrates an efficient processing systems in a recent semiconductor technology with a customer specific set of sensors.

To harden the system against basic tampering, critical security features are embedded deeply into the 3D-SiP. However, advanced physical protection is beyond the scope of this work. Additional functionality is provided through software libraries and Application Programming Interfaces (APIs). This approach aims to reduce the product development costs and, at the same time, increase the overall security of IoT devices.

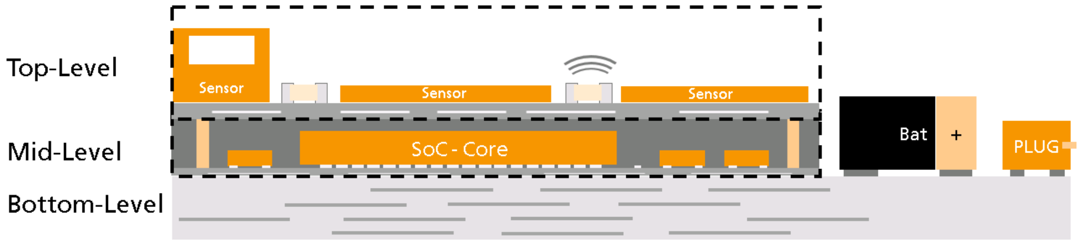

Figure 1. Universal Sensor Platform (USeP) Layer Stack-Up (3D-SiP).

Contributions.

- Discussion of the Universal Sensor Platform (USeP) as flexible computing platform based on the RISC-V ISA (Section II)
- Introduction of the security concept and the required security features (Section III)
- Presentation of the incorporated security mechanisms (Section IV)

II. Universal Sensor Platform

Today's IoT devices can be divided into three groups [4]: low-end, middle, and high-end IoT devices. Middle-end IoT devices can "house more than one communication technologies unlike many of the low-end devices" and may feature "image recognition by running low-level computer vision algorithms". Examples are the ESP8266 or the Samsung ARTIK 05x family. High-end IoT devices on the other hand are mostly single board computers like the Raspberry Pi. Despite the broad range of devices, customization by the system integrator is limited. While it is possible to design a custom PCB for IoT devices this approach limits further miniaturization. "The bottom line, however, is that several electronics devices have reached a near-optimal form factor already and stretching further to get a cutting-edge miniaturization will be very expensive" [4]. The combination of high cost, know-how barriers and long time-to-market hinder SMEs, and especially start-ups, to design their own miniaturized IoT hardware.

The USeP is a platform developed by several Fraunhofer Institutes, which provides a solution for the middle and high-end market. It addresses these issues through two main factors:

- A custom layer stack-up enables a trade-off between scalability and flexibility. The layers enable the combination of a standardized, efficient computation unit with a flexible and adaptive sensor configuration that can be adjusted even at low volumes.
- Automated design processes make custom configurations available without requiring an extensive electronics background or significant design effort.

Potential applications include industrial automation, condition monitoring, smart home, and blockchain applications.

Figure 1 shows a sample composition of the three layers of integration. They differ by the integration density and flexibility for modification.

- *Top-Level (Sensors and Communication):* An individual top-level contains the sensors and communication interfaces that are integrated into one package with the System on Chip (SoC). It permits the design of IoT devices tailored to a specific application. Individualization at this level is more complex and expensive than designing a PCB, but requires less effort than a custom IC or custom package.
- *Mid-Level (SoC Core System)*: The SoC manufactured in 22FDX-technology forms the foundation that remains the same across several different implementations. This level also features some integrated passive devices and connects the levels to each other. In most instances this level will not be adapted for individual customers.
- *Bottom-Level (PCB):* Low-level components that are not integrated in the package (power supply, connectors) are mounted and connected on a PCB. This level can easily be individualized, for example by an SME customer.

The SoC is manufactured in GLOBALFOUNDRIES' 22FDX technology, which offers features favorable for IoT applications like RF connectivity integration, embedded non-volatile memory (MRAM), and low power support (V_{nom} as low as $0.4\,\text{V}$). Due to it being a Silicon-on-Insulator (SOI) technology, it is possible to perform performance/power trade-offs at run-time by adjusting the back-gate bias voltage [5].

The USeP SoC features a RISC-V processor that benefits from the openness of RISC-V, especially in the security context. While open source software is widely available, Open Source Hardware (OSH) is lately also gaining more attention, partly fuelled by recent hardware bugs like Spectre and Meltdown. By OSH we here refer to hardware designs where at least the HDL code is distributed under a license similar to traditional open source software licenses. However, one should be aware that a) RISC-V processors can be OSH but this is no necessity and b) the 'open' usually only covers the HDL Code (front-end design). Both back-end design and fabrication remain proprietary for cutting-edge technology nodes [6].

OSH results in multiple benefits regarding security:

- OSH lowers barriers for security evaluations. If a company plans to use a specific open source core, it can simply perform or assign independent security evaluations.
- OSH attracts research interest which can evolve into communities that continue to enrich the OSH ecosystem with their contributions. For example, there are open pro-

cessor verification tools [7] and hardware fault injection frameworks [8] for RISC-V systems.
- OSH enables custom security extensions, which add registers, cryptographic accelerators, or even new ISA extensions. For example, there is work towards trusted execution environments for RISC-V systems [9].

For our application, we incorporated the following security-related hardware extension: cryptographic accelerators for authenticated encryption and a hash function, a true random number generator, and dedicated internal memory. These security components together with the security concept described in the next sections lay the foundation for the security of the USeP platform.

III. Security Concept

In our security concept, we aim to support solutions that address the three high-level mitigation goals identified in NIST IR 8228 [2]. Each mitigation goal and our corresponding features are detailed in the following sections.

A. *Protect Device Security*

Device security must be ensured over the entire life-time of IoT devices. To support this goal, we focus on protecting the device software.

An important characteristic of IoT devices is that they are interconnected. Networks can be private and separated from the internet, but in general this is not the case. This poses a security risk whenever vulnerabilities are found. IoT devices must include a secure software update mechanism to address this risk.

We provide this functionality through secure boot. It runs at start-up to authenticate the boot image; updates are verified during runtime. Our implementation of secure boot is described in Section IV-A.

It is not sufficient to only provide the device software update functionality. To ensure a quick distribution of configuration changes and updates, a centralized management system must be able to remotely send them to the IoT devices. It must monitor this process to detect failures, for example caused by adversarial systems refusing to update.

We implement this using the authenticated watchdog timer, which is specified as part of the Cyber-Resilient Platforms (CyReP) program [10]. The authenticated watchdog timer is similar to a traditional watchdog timer; it resets the device once its internal timer expires, unless the reset is deferred by software. However, once enabled, it cannot be disabled in the current power cycle and it can only be deferred with deferral tickets issued by an authenticated party.

The central management system, as the authenticated party permitted to issue deferral tickets, is able to recover devices it determines to be misbehaving. Deferral tickets are not sent to these devices to force them to reset. Early boot software, protected by secure boot, is then responsible for recovering the devices into a trustworthy state. This allows compromised devices to be recovered in a predictable time frame, even if they are isolated from the network. The implementation of the authenticated watchdog timer is explained in Section IV-B.

In a large network of IoT devices, it is not usually possible to manually verify every device before granting access to the network. Malicious devices added to the network must be detected automatically to prevent access. Software attestation is used by the central management system to ensure that all devices on the network run the expected software before granting access.

We implement software attestation based on the Device Identifier Composition Engine (DICE) project [11], [12], which is also used as part of the CyReP [10] program. DICE creates an identity for each device by combining a unique device secret with the measurement of the first mutable software layer running on the device and, optionally, critical configuration data. Software running after the first layer is integrated into the identity by cryptographically combining its measurement with the identity. This step is not reversible and overwrites the identity with the new one. Early boot software, protected by secure boot, initializes the DICE identity.

In the next step, the identity is used in software to derive cryptographic keys. This provides the advantage that the derived keys are tied to one specific software configuration. Modifications to software by an adversary change the identity; access to the cryptographic keys is therefore prevented. Similarly, cryptographic keys are also restricted to the correct software layer. Before control is handed to the next software layer, its measurement is included in the DICE identity. It therefore does not have access to the cryptographic keys of the previous layer.

The unique device secret is the base of the DICE identity. With access to it, malicious software would be able to recalculate the identity using modified software measurements to hide itself. To prevent this, the unique device secret is lockable. Access in the current power cycle is prevented once the DICE identity is initialized. The unique device secret and the locking function is described in Section IV-C.

B. *Protect Data Security*

IoT devices collect and use sensitive information like access credentials for communication, encryption keys to protect intellectual property in software, and data collected or generated during operation. They must be protected during run-time and at rest using storage encryption. Similarly, data transmitted over the network must be protected.

The necessary keys for this are derived from the DICE identity (Section III-A) with a Key Derivation Function (KDF). The identity is based on the chip-unique cryptographic key provided by USeP (Section IV-C). This ensures that data and communication remain secure, even if the software on the device is modified by an adversary.

The keys derived from the DICE identity are, in addition, protected across factory resets. Early boot software adds a user identity into the DICE identity to personalize it. The first software layer is started after this step is complete. At every factory reset, a new random user identity is generated. This

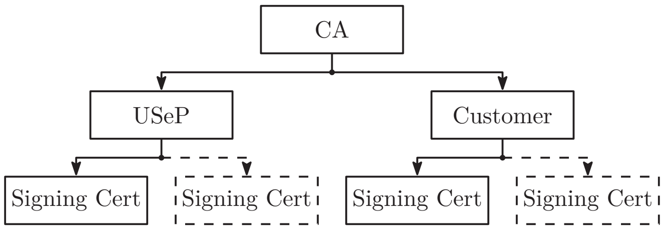

Figure 2. Certificate chain used in the USeP platform. Signing certificates added at a later point are indicated with dashed lines.

ensures that all cryptographic keys used by the system will change with factory resets.

C. *Protect Individuals' Privacy*

IoT devices are designed to collect information on their environment, analyze it, and then display it or act directly on the results. This information is often supplemented with manually entered data to improve the system. Together, this allows conclusions to be drawn about the user and therefore represents a privacy risk. Care must be taken to prevent unauthorized access to it.

This goal is achieved with policies restricting the flow and processing of Personally Identifiable Information (PII), and by minimizing the need for PII [2]. This is highly dependent on the use-case of the final system. As platform, USeP implements features to protect the device and data security to, in turn, protect PII on the device. However, it does not provide hardware features specific to only this goal.

IV. Security Mechanisms

This section describes the specific methods and functions to implement the introduced security concept.

A. *Secure Boot*

Secure boot [13], [14] is a secure bootstrap method for processors. It requires software components to be authenticated before being allowed to execute. Software is authenticated by verifying software signatures or by comparing its hash digest with a signed list of approved software components. If a component with an invalid signature or checksum is encountered, the secure boot process is stopped.

Typically, the device manufacturer determines what software is approved. In the case of USeP, we have two entities, who control what software is approved to run on the device, the USeP organization and the customer integrating USeP into their product. Access to the machine-mode is only granted to software approved by the USeP organization; customer software is restricted to the user-mode. To handle this use-case, two certificate slots are available to store the public signing certificates of both entities.

The certificate chain used by our secure boot implementation is shown in Fig. 2. Only the USeP and customer nodes are stored in the certificate slots. Both have one or more signing certificates. They are transferred with the software image and are used for software verification. The Certificate

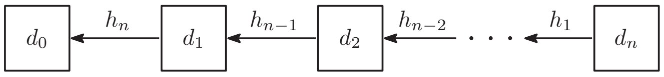

Figure 3. Block diagram of the authenticated watchdog timer hash chain.

$$\begin{tikzcd} d_0 & d_1 \arrow[l, "h_n"'] & d_2 \arrow[l, "h_{n-1}"'] & \cdots \arrow[l, "h_{n-2}"'] & d_n \arrow[l, "h_1"'] \end{tikzcd}$$

Authority (CA) is stored on the device; devices may be configured without the CA during wafer production, creating two independent certificate chains.

To support certificate revocation, each signing certificate has a monotonically incrementing serial number. Locally, in internal non-volatile storage, USeP stores the lowest-permitted serial number. During verification, the serial number of each certificate is compared with it to determine if it was revoked. Certificates can only be revoked using a firmware function, to limit what the lowest-permitted serial number may be set to. To prevent downgrade attacks, it cannot be decremented. To prevent denial of service attacks, it cannot be incremented past a set offset from the serial number of the installed signing certificate — the certificate software requesting the revocation is signed with.

ROM firmware is responsible for starting the secure boot process. It, first, verifies and installs the signing certificates from the software image. It then authenticates and starts the machine-mode software from the software image. Only software signed by a signing certificate of the USeP organization is permitted to run at this stage.

The boot process is continued by the machine-mode software. It verifies and starts the user-mode software from the software image. Software signed by certificates belonging to the USeP organization or the customer is permitted to run at this stage. The installed certificates may also be used to verify updates for the device.

B. *Authenticated Watchdog Timer*

The authenticated watchdog timer enables control of the device, even if an adversary has gained access to it. It resets the device once its timeout interval has elapsed and it has not been deferred. Deferral is only possible with deferral tickets sent by an authenticated external party, the central management system. Thus, if the external party detects irregular behavior, it stops sending deferral tickets to force a reset of the device. Control is regained by the early boot software, which is responsible for recovery.

The authentication protocol of the authenticated watchdog timer is based on Lamport's one-time password scheme [15]. This scheme is secure from an attacker eavesdropping on the communication between the client and the server. Passwords sent by the client to authenticate itself are hard to guess for the attacker, even with knowledge of previous passwords, and are one-time useable.

Lamport's scheme [15] uses hash chains to generate the one-time passwords. A hash chain represents the iterative application of a hash function $h(x_{i-1}) = x_i$, with the initial input x_0. Each iteration outputs one one-time password x_i, where i corresponds to the i^{th} iteration of the hash function $h(x)$.

The concept for this authentication scheme was developed and standardized as S/Key [16]. Kogan, Manohar, and Boneh [17] extended S/Key to create the time-based one-time password system T/Key. We base our implementation of the authenticated watchdog timer on T/Key, simplified and adjusted to our application.

T/Key [17] is a second-factor authentication system. Individual passwords are typically valid for 30 seconds. Their hash chains are long — typically 2×10^6 entries for a hash chain valid for 2 years — to avoid frequent reinitialization. In our use-case, the passwords, or deferral tickets, have a longer validity period to reduce traffic to the IoT devices; in most cases 24 hours or longer. As a result, the size of the hash chain is significantly smaller; it does not use the checkpointing system of T/Key for efficient hash chain traversal.

Compared with S/Key [16] and Lamport's scheme [15], T/Key [17] uses n independent hash functions h_i for a hash chain of length n. This removes the dependence of the security on the length of the hash chain. The hash functions h_i for $1 \leq i \leq n$ are defined as

$$h_i(x) = H(n - i \parallel s \parallel x),$$

where $H(\cdot)$ is a hash function, s a salt, and $a \parallel b$ denotes the concatenation of a and b. All three inputs to the hash function $H(\cdot)$ are of fixed length. Compared with T/Key, we use the relative hash chain index i instead of the absolute index in terms of the time of the passwords' validity.

The authentication protocol of the authenticated watchdog timer between the central management system and a device works as follows:

1) Setup: The central management system — the client in T/Key [17] — begins the setup by generating the cryptographic key k and the salt s for the device — the server in T/Key. It then calculates the first deferral ticket

$$d_{\text{init}} = d_0 = h_n(h_{n-1}(\dots h_1(k) \dots)),$$

which corresponds to the last element in the hash chain with length n and the initial value $d_n = k$. Both are shown in the simplified hash chain in Fig. 3. It sends d_{init} and salt s to the device, which stores them as d_{prev} and s respectively. In addition, both the central management system and the device keep track of the hash chain index, the central management system as the index j of the next deferral ticket and the device as the index j_{prev} corresponding to d_{prev}.

This step is repeated after all deferral tickets of the hash chain have been depleted, causing a device reset. If the values stored by the device are not persisted over power cycles, setup must be completed with every device start.

2) Authentication: During authentication, the central management system calculates the next deferral ticket d_j at the hash chain index j:

$$d_j = h_{n-j}(h_{n-j-1}(\dots h_1(k) \dots)).$$

The last deferral ticket, at the hash chain index $j = n$, corresponds to $d_n = k$. No new deferral tickets can be generated at this point; the device is reset once the watchdog timer expires. The setup must be repeated as part of the next device boot.

The device verifies the deferral ticket by calculating the one preceding it,

$$d'_{\text{prev}} = h_{n-j_{\text{prev}}}(d_j).$$

If d_{prev} and d'_{prev} match, authentication is successful and the authenticated watchdog timer is deferred. The device changes d_{prev} to d_j, and updates the hash chain index j_{prev}. Otherwise, the authenticated watchdog timer is not deferred and the device will be reset once it expires. Note that deferral tickets received by the device are always directly preceded by the stored last deferral ticket d_{prev}.

3) Implementation: The implementation consists of both software and hardware components. Software is responsible for communication between the device and the central management system; the timer, and deferral ticket verification is implemented in hardware on the device.

The hardware interface of the authenticated watchdog timer consists of four registers: d_{prev}, d_j, j_{prev}, and s. The register names correspond to the variables used in Sections IV-B1 and IV-B2. An additional control and status register is used to control the authenticated watchdog timer.

Registers d_{prev}, j_{prev}, and s are readable and writable by software while the authentication watchdog timer is inactive and turn read-only once active. Software must configure them with the values obtained during setup.

Register d_j is always readable and writable by software. It is populated with the deferral ticket by software as part of the authentication step.

Hardware is still able to write registers d_{prev} and j_{prev}. On success authentications, the value in d_j is copied into d_{prev} and the value in j_{prev} is updated.

C. Key Management

Key management on USeP consists of a chip-unique key and the key Control and Status Registers (CSRs). The chip-unique key is a $256\,\text{bit}$ cryptographic key, which is generated internally during manufacturing. It is stored in internal One-Time Programmable (OTP) memory, which is not directly accessible by software. It is only available using the key CSR.

The key CSR provides an interface for accessing cryptographic keys on the platform. It is both readable and writable by machine-mode software. The chip-unique key is made available, once the device state is determined to be secure by secure boot. Similarly, if software determines that a security violation has occurred, the key CSR is zeroized by software to prevent access to the chip-unique key.

The chip-unique key is used as the unique device secret required to construct the DICE identity [11]. It is locked, by overwriting the key CSR with the initial DICE identity, preventing any further access to it in the current power cycle. All further additions to the identity are also stored in the key CSR, thereby removing access to keys derived from an earlier DICE identity.

V. Conclusion and Outlook

In this work, we discuss a security architecture for RISC-V based IoT devices. It provides the required features to achieve the three risk mitigation goals identified in NIST IR 8228 [2]: protect device security, data security, and individuals' privacy.

The security architecture focuses on three areas: secure boot, the authenticated watchdog timer, and key management. Secure boot ensures that only authenticated software is permitted to run on the device. Software is verified as part of the boot process, but also during software updates. To ensure software updates are applied, we implement the authenticated watchdog timer specified in the CyReP program [10]. It allows an external party to force a reset of misbehaving devices to enable recovery by early-boot software. The platform provides a device-unique key, intended to protect data security. It is used to implement the DICE project [11], [12], which creates a software identity for the device. This enables software attestation and allows keys to be tied to a valid software state.

The security architecture will be implemented as part of the USeP project. The SoC will be manufactured in GLOBALFOUNDRIES' 22FDX technology and aims to give SMEs access to state-of-the-art technology for IoT devices.

VI. Acknowledgements

The authors would like to thank Klaus Meyer, Joachim Haase, Ludger Irsig, and Andreas Brüning for the discussions and feedback. The project is funded by the European Union and the Free State of Saxony as part of the European Regional Development Fund (ERDF).

References

[1] K. Rose, S. Eldridge, and L. Chapin, "The Internet of Things (IoT): An Overview", The Internet Society, Oct. 2015. [Online]. Available: https://www.internetsociety.org/wp-content/uploads/2017/08/ISOC-IoT-Overview-20151221-en.pdf.

[2] K. Boeckl, M. Fagan, W. Fisher, N. Lefkovitz, K. Megas, E. Nadeau, B. Piccarreta, D. Gabel O'Rourke, and K. Scarfone, "Considerations for Managing Internet of Things (IoT) Cybersecurity and Privacy Risks", National Institute of Standards and Technology, NISTIR 8228 (Draft), Sep. 24, 2018. DOI: 10.6028/NIST.IR.8228-draft.

[3] zGlue. (Nov. 28, 2018), [Online]. Available: https://zglue.com/technology.

[4] M. O. Ojo, S. Giordano, G. Procissi, and I. N. Seitanidis, "A Review of Low-end, Middle-end and High-end IoT Devices", *IEEE Access*, pp. 1–1, 2018. DOI: 10.1109/ACCESS.2018.2879615.

[5] GLOBALFOUNDRIES. (2018). Product Brief 22FDX FD-SOI Technology, [Online]. Available: https://www.globalfoundries.com/sites/default/files/product-briefs/pb-22fdx.pdf.

[6] G. Gupta, T. Nowatzki, V. Gangadhar, and K. Sankaralingam, "Kickstarting Semiconductor Innovation with Open Source Hardware", *Computer*, vol. 50, no. 6, pp. 50–59, 2017. DOI: 10.1109/MC.2017.162.

[7] J. Choi, M. Vijayaraghavan, B. Sherman, A. Chlipala, and Arvind, "Kami: A platform for high-level parametric hardware specification and its modular verification", eng, *Proceedings of the ACM on Programming Languages*, vol. 1, no. ICFP, pp. 1–30, 2017. DOI: 10.1145/3110268.

[8] S. Eldridge, A. Buyuktosunoglu, and P. Bose, "Chiffre: A Configurable Hardware Fault Injection Framework for RISC-V Systems", in *Second Workshop on Computer Architecture Research with RISC-V (CARRV)*, 2018.

[9] V. Costan, I. A. Lebedev, and S. Devadas, "Sanctum: Minimal Hardware Extensions for Strong Software Isolation", in *USENIX Security Symposium*, 2016, pp. 857–874.

[10] A. Marochko, D. Mattoon, P. England, R. Aigner, R. Spiger (CELA), and S. Thom, "Cyber-Resilient Platforms Overview", Sep. 2017. [Online]. Available: https://www.microsoft.com/en-us/research/publication/cyber-resilient-platforms-overview/.

[11] Trusted Computing Group, "Device Identifier Composition Engine (DICE) Architectures". [Online]. Available: https://trustedcomputinggroup.org/work-groups/dice-architectures/.

[12] P. England, A. Marochko, D. Mattoon, R. Spiger, S. Thom, and D. Wooten, "RIoT - A Foundation for Trust in the Internet of Things", Microsoft Research, Apr. 2016. [Online]. Available: https://www.microsoft.com/en-us/research/publication/riot-a-foundation-for-trust-in-the-internet-of-things/.

[13] W. A. Arbaugh, D. J. Farber, and J. M. Smith, "A Secure and Reliable Bootstrap Architecture", in *IEEE Symposium on Security and Privacy*, IEEE, 1997, pp. 65–71.

[14] M. Gasser, A. Goldstein, C. Kaufman, and B. Lampson, "The Digital Distributed System Security Architecture", National Institute of Standards and Technology, Jan. 1989.

[15] L. Lamport, "Password Authentication with Insecure Communication", *Commun. ACM*, vol. 24, no. 11, pp. 770–772, Nov. 1981. DOI: 10.1145/358790.358797.

[16] N. Haller, "The S/KEY One-Time Password System", Internet Engineering Task Force, RFC 1760, Feb. 1995. DOI: 10.17487/rfc1760.

[17] D. Kogan, N. Manohar, and D. Boneh, "T/Key: Second-Factor Authentication From Secure Hash Chains", in *ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17, ACM, 2017, pp. 983–999. DOI: 10.1145/3133956.3133989.