IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN XXXX-XXXX, Vol. 0, No. 0, pp. 1–24. DOI:XXXXXXXX

BASTION: A Framework for Secure Third-Party IP Integration in NoC-based SoC Platforms

Francesco Restuccia[1], Zhenghua Ma[2], Joseph Zuckerman[3], Andres Meza[4], Biruk Seyoum[5], Luca Carloni[6], and Ryan Kastner[7]

[1] University of California San Diego, La Jolla, California, USA, frestuccia@ucsd.edu
[2] University of California San Diego, La Jolla, California, USA, zhm007@ucsd.edu
[3] Columbia University, New York City, New York, USA, jzuck@cs.columbia.edu
[4] University of California San Diego, La Jolla, California, USA, anmeza@ucsd.edu
[5] Columbia University, New York City, New York, USA, biruk@cs.columbia.edu
[6] Columbia University, New York City, New York, USA, luca@cs.columbia.edu
[7] University of California San Diego, La Jolla, California, USA, kastner@ucsd.edu

Abstract. Modern System-on-Chip (SoC) architectures are a complex mix of processors, accelerators, memories, and I/O controllers interconnected by on-chip communication networks. Given the complexity of the computation and the requirements mandated in modern applications, several of these IPs are often outsourced as third-party modules. The integration of third-party modules, however, has been demonstrated to raise severe system-level security concerns – undiscovered vulnerabilities, incorrect firmware configurations, malicious code, and hardware trojans undetected in such IPs can produce leaks of confidential information and compromise the integrity of critical components. These challenges are further intensified when the communication infrastructure lacks robust mechanisms to supervise and monitor the interactions of third-party IPs with the rest of the system. Thus, runtime monitoring and supervising of third-party IPs is a crucial aspect for the system-level security of the entire SoC – the computing modules integrated in the SoC and their communication must behave securely. This paper presents Bastion, an open-source framework designed to support the secure integration of third-party IP modules into SoC architectures based on network-on-chip (NoC) communications, with a focus on providing robust security guarantees for NoC-based open-source hardware platforms. Unlike most previous works, which either focus on design or verification, we address the challenge of securely integrating third-party IPs in NoC-based platforms through a holistic design and verification framework based on three pillars: (i) a high-performance security socket that can be seamlessly integrated into NoC tiles; (ii) secure configuration and management of the security sockets via a Hardware Root of Trust; and (iii) an ad-hoc property-based security verification framework to ensure secure system operation. Bastion is integrated on the popular open-source ESP framework and validated through simulations and FPGA emulation of realistic SoCs. By explicitly targeting open-source platforms and releasing the entire project as open-source, we aim to democratize access to robustly secure application-specific SoC platforms for critical applications and foster further advancements in this domain.

Keywords: System-on-Chip · NoC-based platforms · Access Control Systems

1 Introduction

System-on-chip (SoC) architectures incorporate tens to hundreds of heterogeneous hardware components, including processors, accelerators, cryptographic cores, a Hardware Root of Trust (HWRoT), on-chip memories, DMA engines, and I/O controllers. Given the

complex performance requirements of modern applications, these modules are typically highly specialized – outsourcing and reuse of third-party IPs developed by specialized external companies or open-source communities has become a standard practice to reduce design complexity and time-to-market. The integrated IP modules require an on-chip communication infrastructure to exchange information and coordinate tasks. While most traditional SoC interconnects rely on crossbar-based solutions, modern SoCs increasingly adopt Network-on-Chip (NoC) communication infrastructures to manage on-chip communications [KNRSV00].

Memory-intensive third-party IPs often require direct physical access to peripherals in order to achieve high performance. A relevant example is hardware accelerators for AI applications, which need substantial bandwidth to retrieve weights and input data from DRAM memories. While direct memory access provides significant performance benefits, it also poses significant security threats in the presence of third-party IPs – backdoors, design flaws, or malicious configurations in these IPs can compromise and enable unauthorized access to sensitive data if access control mechanisms at the on-chip communication infrastructure are not present or insufficiently robust.

In these scenarios, secure on-chip access control is pivotal for preserving data integrity and confidentiality. Unfortunately, according to the hardware Common Weakness Enumeration (CWE), access control is one of the largest categories of hardware flaws, encompassing roughly one-fifth of all identified weaknesses [MIT]. This underscores the need for a solution that not only enforces effective security policies, but can also be shown to be free of known weaknesses that would allow malicious IPs to compromise system confidentiality or integrity, especially in critical SoC applications including automotive, avionics, robotics, and medical applications.

Robust verification frameworks are increasingly being used to validate both the correctness of access control lists and their hardware implementations [KRM+22], helping detect weaknesses such as misconfigurations and logical errors that could otherwise lead to unauthorized data exposure or corruption. Nevertheless, existing approaches for securing NoC-based platforms generally concentrate on either design [PGC22, FPL+08] or verification [SAHS+18], leaving a gap for a more holistic approach that combines hardware design, security verification, and secure management of access control lists. This is especially problematic for open-source NoC-based platforms, which can become vulnerable to access-control threats in the presence of unverified, third-party IP modules.

1.1 Contribution

We propose Bastion – a framework for the development, deployment, and security verification of access control infrastructures targeting the challenges of NoC-based SoC architectures. Bastion's main goal is to provide a holistic design and verification solution to enforce verifiable security features. We aim to achieve this goal with particular attention to enhancing security in open-source SoC platforms. Hence we developed, integrated, and evaluated Bastion on the popular ESP open-source platform for heterogeneous SoC design [MGDG+20]. In particular, our contributions are:

- *Design:* an open-source security socket (sSocket), integrated into SoC tiles and interfaced to the NoC, which provides runtime-configurable isolation and access control services. The sSocket is transparent to the integrated IPs and thus compatible with any third-party IP supported by the NoC backbone[1];
- *Security verification:* a property-based security verification flow that ensures secure operation of the proposed security infrastructure against the MITRE CWEs at the tile, communications, and system level, and extensible to cover evolving threats;

[1] https://github.com/KastnerRG/ESP-Bastion

- *Secure configuration:* a methodology for the secure configuration and management of the security sockets from an open-source HWRoT, ensuring secure configuration through trusted communications;
- *Support for secure open-source platforms:* the integration and verification of our proposed methodology on the open-source ESP platform, validated with RTL simulations and FPGA implementations on realistic SoC designs.

Table 1 reports a summary comparison of Bastion with the most related academic and industrial solutions. Most research contributions on NoC-based platforms focus either on design or verification, missing a holistic design and security verification framework capable of providing securely verified access control features. Existing proprietary hardware solutions supporting software Trusted Execution Environments at the system-level [ARM] miss crucial aspects in the open-source context to ensure trustiness and extendibility, including security verification transparency and design availability.[2] Bastion aims to bridge this gap, providing a baseline framework for the development and maintenance of secure and verifiable open-source SoC platforms, supporting trust and transparency in critical domains. We summarize features, limitations, and future works of our approach in Section 7.

Table 1: Summary comparison of Bastion with the state of the art. Section 3 reports a detailed description of the related works.

Approach	NoC-based socket design	NoC-based security verification	Open source release	Dynamic reconfiguration	Verified secure configuration
AKER	No	No	Yes	Yes	Yes
Sepulveda et al.	No	Yes	No	No	No
Karabalut et al.	No	No	No	Yes	No
Grammatikakis et al.	Yes	No	No	Yes	No
Fiorin et al.	Yes	No	No	Yes	No
Commercial TEEs	Yes	No	No	Yes	No
Bastion	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**

1.2 Paper outline

The rest of the paper is organized as follows: Section 2 provides the context and motivation for this work. Section 3 reports the related works. Section 4 describes Bastion's security socket (sSocket) design, its integration within the ESP platform, and discusses the generalization of integration in other platforms. Section 5 describes Bastion's security verification framework. Section 6 reports our experimental campaign, conducted on multiple SoC designs both in simulation and on realistic use-case scenarios deployed on FPGA platforms. Section 7 reports a brief discussion of features and current limitations of our approach and future works. Finally, Section 8 concludes the paper.

2 Context and Motivation

2.1 NoC-based SoC architectures

The complexity of computations required in modern SoCs demands the use of highly specialized computing units and resources to achieve the performance necessary in current

[2]it is worth noting that whenever required, Bastion can also be leveraged to enforce system-level isolation across the NoC shared resources in a proprietary software TEE environment setting)

applications. These modules are often outsourced to third-party vendors to reduce development complexity. In this context, modern SoCs are composed of multiple hardware module resources that can be categorized into *controllers* (e.g., processors and accelerators) and *peripherals* (e.g., memories and I/Os). Each hardware module acts as a controller or peripheral (or both). In NoC-based architectures, resources are integrated into *tiles*. Tiles are typically organized in a mesh configuration. Each tile contains a *router* for communication with neighboring tiles. During execution, controllers actively submit read/write requests to peripherals. Requests are collected by the local router of the source tile and propagated through the NoC until reaching the destination tile. The peripheral in the destination tile serves the transaction, either accepting write data from controllers or responding with the requested data. Peripherals are memory-mapped and often shared among multiple controllers.

The ESP framework [MGDG+20] is a widely used open-source project that facilitates custom NoC-based heterogeneous SoC development, adopting a third-party IP reuse approach to accelerate development. ESP provides a robust NoC backbone that includes security services such as address translation units for accelerators developed using the ESP flow. ESP focuses on the design and integration of custom and third-party IP modules into its SoC tile-based architecture, which is based on a NoC with a 2D-mesh topology. Each IP module is integrated into a tile, while the interface with the NoC is automatically generated and provides specific platform services to the IP modules. This creates compatibility and flexibility for the out-of-the-box integration of several third-party IPs, whether commercial, open-source, or internally developed.

While ESP provides a robust and stable backbone for communications among the tiles, it does not necessarily consider the integrity and confidentiality of the data transfers among resources – for enhanced compatibility, ESP allows direct access to the entire physical address space in its third-party integration flow. This feature ensures out-of-the-box compatibility for third-party IP software drivers. However, it also introduces significant risks by granting unsupervised access to the entire SoC address space, exposing the system to potential integrity and confidentiality threats. This raises significant security concerns not only within the third-party IP itself, but also at the system level, as witnessed by prior works [RPB+19] and by the MITRE hardware Common Weakness Enumerations (CWEs) database [KRM+22].

2.2 Access control systems in NoC-based platforms

An essential aspect of critical systems is preserving runtime data integrity and confidentiality. Untrusted third-party controllers capable of issuing requests to shared resources must be proactively monitored to prevent malicious attacks, such as Direct Memory Access (DMA) attacks [TL17]. Employing zero-trust principles ensures that third-party IPs can only access the resources necessary for their task, mitigating risks of data leakage or unauthorized access. A robust control system that guarantees practical and verifiable secure runtime access to system resources by third-party IPs is crucial. Several methodologies focusing on providing robust security guarantees in NoC-based platforms are available in the state of the art, either focusing on security verification [SAHS+18, ZRMH+24, KRM+22] or design methodologies [RCSP23, FPL+08]. However, an open-source holistic access control system framework for NoC-based architectures combining design, secure configuration, and rigorous security verification supporting proactive monitoring and supervision of the runtime behavior of third-party IPs is notably absent, leaving many current open-hardware platforms, including ESP, vulnerable to serious security threats.

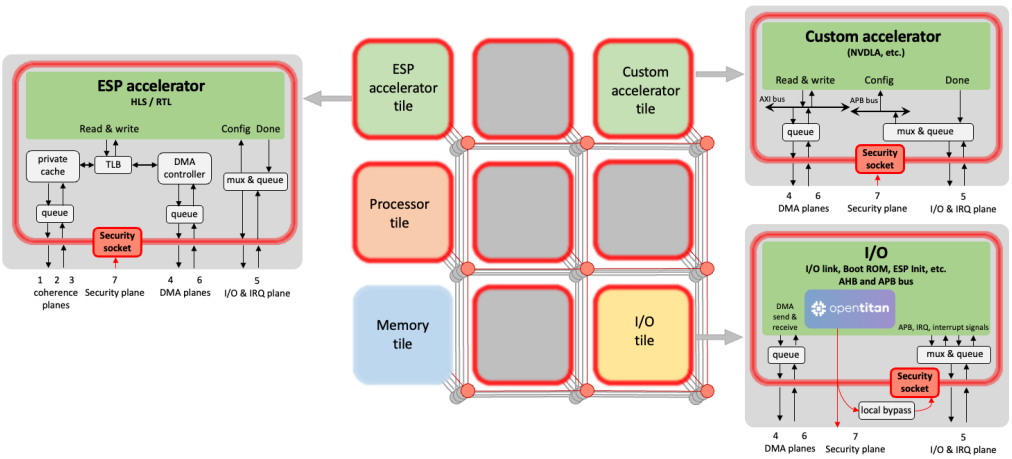

Figure 1: BASTION deployed on a sample 3x3 ESP architecture.

2.3 Threat model

We consider a typical SoC development flow that involves integrating one or more third-party IPs into the system, leveraging the ESP workflow for SoC development. We assume that these IPs may originate from external vendors or open-source repositories, making their internal behavior and security guarantees difficult to validate comprehensively. The IPs may have a different level of trustworthiness – thus, each IP must have minimal access to the shared resources. The ability of untrusted third-party IPs to have unsupervised access to the resources in the system is a threat to the system's integrity and confidentiality. This principle ensures that each IP can exclusively access the resources strictly necessary for its operation. BASTION enforces the access control list on the untrusted IP modules – an access control list defines each IP's access rights to shared resources, which must be enforced to preserve the system's security. Access control lists (ACLs) provide a mechanism to specify these rights, defining precise permissions at the level of individual IP modules. Our threat model focuses on evaluating the security of the BASTION infrastructure. Thus, we assume the NoC routers to be trusted – the security of the NoC backbone is beyond the scope of this work and can be ensured, for instance, by applying the work proposed in [SAHS+18].

We integrate a trusted entity, e.g., a HWRoT, managing the access control list (see Section 4.3). The HWRoT serves as the cornerstone for secure policy configuration and enforcement, orchestrating the programming of ACLs into the security sockets. Considering the documentation of most modern open-source HWRoT, we assume it to be resilient against tampering, side-channel attacks, and privilege escalation. Advanced security verification on the HWRoT is beyond the scope of this work and can be assured, for instance, by applying the methodologies proposed in [MRO+23]. We provide properties to verify that the trusted entity correctly programs and maintains the access control lists in the security sockets (Section 5). These properties ensure the secure operation of the access control enforcement. Currently, BASTION focuses on system confidentiality and integrity. Denial-of-service [RK22] and flood attacks of legal transactions [RPB+19] are excluded from the threat model and beyond the scope of this paper. As discussed in Section 7.1, our future works aim to expand BASTION's functionalities ainlso in this direction.

3 Related Works

3.1 Security verification

Security verification is fundamental to providing robust guarantees of a system's security. Several techniques have proven highly effective in addressing a wide range of threats. These include methodologies targeting side-channel attacks [ZCF24], automating the verification of physical security properties to mitigate attacks such as information leakage and tampering [RBFSG22], and identifying and addressing critical security flaws in pre-silicon designs of security-critical hardware modules like the OpenTitan Hardware Root of Trust [MRO+23]. Provably verifiable security features are a fundamental requirement in modern security-critical systems, as witnessed by the ever-increasing number of hardware weaknesses found and exploited in commercial systems [KRM+22]. These vulnerabilities emphasize the critical need for frameworks that integrate both security features and rigorous verification methodologies, particularly in scalable and modular SoC architectures.

In the context of NoC architectures, Sepulveda et al. [SAHS+18] proposed a methodology describing the correct behavior of NoC routers through security properties. This approach provides a strong foundation for verifying the integrity of NoC routers but does not address system-level access control, which is the focus of Bastion. Similarly, Siddiqui et al. [SHS18] and Tan et al. [TEF+20] proposed methodologies for supporting the detection of anomalous conditions during the execution of modules.

These works make significant contributions to security verification, but they do not provide a methodology for developing secure access control infrastructures in conjunction with the proposed security verification. Bastion fills this gap by combining hardware design with property-based security verification and dynamic management capabilities, aiming at offering a comprehensive approach to secure hardware development.

3.2 Access control systems design

Access control in SoCs has been deeply investigated across different architectures. Huffmire et al. [HPSK06] and Brunel et al. [BPOD14] proposed mechanisms for securing off-chip memories. Restuccia et al. [RMKO22, RMK21] and Cotret et al. [CCGD12] developed solutions for access control systems on crossbar-based architectures. Karabalut et al. [KAA23] and Malik et al. [MKAA24] proposed access control systems targeting resource efficiency in multi-tenancy cloud FPGA systems. Rodriguez et al. [RCSP23] recently introduced an I/O Memory Management Unit (IOMMU) compliant with the RISC-V specification. These works address access control for crossbar-based systems, which require distinct considerations for scalability and modularity compared to the NoC-based systems targeted by Bastion.

Considering NoC-based platforms, Piccolboni et al. [PGC22] described the concept of a security socket in NoC-based platforms, focusing on providing static security services at the IP level within the tile. Grammatikakis et al. [GPP+15] and Fiorin et al. [FPL+08] proposed access control systems for NoC-based architectures. These solutions share similar objectives with Bastion's sSocket design, particularly in their aim to enforce access control lists in NoC-based systems. Given the critical importance of access control for system security and the significant number of hardware weaknesses related to access control identified by the MITRE consortium, we identify security verification as fundamental for practical applicability in realistic critical applications. In Bastion, our contribution complements a design methodology with a rigorous security verification methodology offering robust guarantees for the developed security infrastructure and the integration into an open-source and widely-used NoC architecture.

3.3 Open-source hardware platforms

Open-source SoC platforms have reached an impressive level of maturity. The PULP project provides several application-specific SoCs, including those designed for AI applications at the edge [CGR+24], chiplet-based High-Performance Computing [SBP+25], and extended reality applications [PSC+24]. The ESP framework [MGDG+20] offers a robust environment for the development of NoC-based SoC platforms, emphasizing the concept of bring-your-own-IPs [GCE+20]. This facilitates the seamless integration of third-party IPs by providing a NoC backbone for communication and integration. Chipyard [ABG+20] is another prominent open-source framework, specifically for the development of RISC-V-based SoCs. It leverages the Chisel hardware description language and supports the integration of MMIO-mapped peripherals and custom accelerators, enabling flexible and scalable SoC designs.

Despite their maturity and extensive adoption, security verification remains underexplored in these platforms. The lack of rigorous security verification workflows creates potential vulnerabilities, as hardware security threats may go unnoticed or remain unmitigated. This oversight could have serious consequences, especially in critical applications where security is paramount. Notably, Rogers et al. [RSM+24] recently proposed a set of SystemVerilog Assertions properties targeting common open-source designs, including Hack@DAC 2018's buggy PULPissimo SoC, Hack@DAC 2019's CVA6, and Hack@DAC 2021's buggy OpenPiton SoCs. In this context, Bastion focuses on providing design and verifiable security properties for open-source platforms. These contributions aim to encourage the adoption of open-source platforms in critical domains such as automotive systems, medical devices, and industrial control, ensuring that such platforms meet the rigorous security demands of these applications.

4 Bastion: The Security Socket (sSocket)

4.1 Overall architecture

The architecture of a Bastion-based security system comprises a set of Bastion security sockets (sSockets) that are distributively deployed between the NoC backbone and the untrusted IPs within the tiles. Access to the NoC backbone from the resources in a tile is proactively monitored and controlled by the local Bastion security socket. Each sSocket includes a configuration port through which it communicates with the Hardware Root of Trust (HWRoT), the trusted entity responsible for configuring and managing the access control lists in each sSocket, thereby defining the accessible regions.

Figure 1 showcases the application of Bastion on a sample ESP architecture – each ESP tile is protected with the Bastion security socket (detailed in Section 4.2), defining a local access control list for the transactions issued by the local resource integrated into the tile and directed to the NoC backbone. The security sockets are configured and managed by the HWRoT integrated into the ESP I/O tile (Section 4.3). The communications issued by the HWRoT for the configuration and management of the security sockets are enabled by a secure NoC data plane (as detailed in Section 4.4). The secure operation of Bastion is supported by the property-based security verification discussed in Section 5.

4.2 Design

The security Socket (sSocket) is a configurable security module to be placed in any untrusted NoC tile to safeguard against unauthorized or malicious attempts to access shared resources. For optimal compatibility, the sSocket operates directly on raw NoC flits, ensuring protocol independence and enabling seamless integration with a wide range of third-party IPs. This abstraction ensures that our approach remains agnostic to the specific

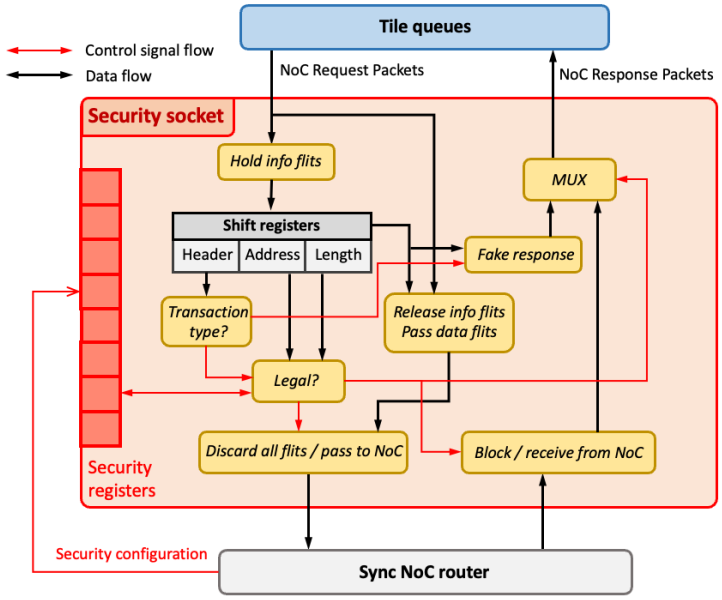

Figure 2: BASTION sSocket structure. The security registers are configured by the HWRoT.

communication protocols used by the IPs integrated into the tiles. Furthermore, this low-level operation enhances the reusability of BASTION in generic NoC-based architectures by positioning the sSocket between the NoC backbone and the local router, making it seamlessly integrable with other NoC-based backbones.

Considering the ESP framework leveraged in this work, the sSocket is positioned between the tile queues and the local NoC router. This design ensures modular integration of sSocket into ESP and requires minimal modifications to the rest of the backbone.

Figure 2 shows the structure of the sSocket. Security registers store the allowable read and write regions for the local IP. Regions are configured and dynamically maintained by the HWRoT, according to the current operative mode of the system. All NoC requests to shared resources are validated by the sSocket before entering the NoC backbone. By operating proactively, the sSocket halts any illegal traffic at the source, ensuring that only legitimate traffic enters the network.

When a request is issued, the sSocket analyzes the header flit to determine the legality of the transaction request; the request is then deemed to be legal or illegal. If the target address does not fall into an allowable region defined in the local access control list stored in the sSocket registers, the request is deemed illegal. Illegal requests are aborted before entering the NoC, ensuring that only legal requests can enter the NoC backbone. In parallel, a protocol-compliant error response is generated and propagated to the local IP issuing the request. An interrupt is sent to the HWRoT via the security plane to notify it about the illegal access attempt. If a request is instead deemed legal, it is released by the sSocket, entering the NoC backbone. For reads, sSocket saves the information about the granted requests and enables the corresponding response flits to enter the local tile.

Whenever necessary, the sSocket can be configured to put the local tile under isolation, blocking all of the transactions issued by the local IP and the transfer of data to the tile. This feature supports critical operations requiring a high degree of isolation, such as system bootup or updates.

4.3 Configuration and management

The flexibility of Bastion relies on the configurability of the sSockets, which requires a trusted entity to manage this process during boot time and subsequently throughout the system's lifecycle, especially in scenarios where updates are needed.

Bastion utilizes the HWRoT for configuration, leveraging proven solutions like OpenTitan [Opeb]. OpenTitan is a mature, open-source HWRoT with extensive documentation, robust verification support, and successful demonstrations in multiple commercial tape-outs, making it an ideal candidate for secure initialization and configuration in Bastion.

In the ESP platform, the IO tile is already equipped with a privileged initialization module responsible for system bootup and hard resets. This module provides an optimal integration point for the trusted entity required by Bastion. By embedding the HWRoT functionality in this tile, the initialization process can securely configure the sSockets and ensure their proper operation without introducing unnecessary complexity or altering the platform's existing bootup procedures. Generalizing this approach to other NoC architectures, a specific tile can be designated as the *security tile* hosting the trusted entity. This tile acts as the central authority for secure initialization and configuration in the system, ensuring compatibility with various NoC-based designs.

To align with the principles of a zero-trust architecture, Bastion minimizes the levels of trust assigned to system components. Aside from the HWRoT and the ESP bootup module, all other IPs integrated into the IO tile are considered untrusted. These components are monitored and supervised by sSockets, ensuring robust oversight and limiting the potential impact of any compromised or malicious IP modules. This strategy enhances the security of the system while maintaining flexibility and scalability across various applications.

Bastion allows both static and dynamic configuration scenarios – static configuration is set at boot time and cannot change at runtime; dynamic configuration allows runtime adjustments to accommodate scenarios in which the target SoC has multiple execution modes. Dynamic configurations are particularly advantageous for adaptive systems that must transition between different operational states, such as performance-optimized modes or during critical operations like firmware updates. To make an example, during a firmware update, the HWRoT can decide through a dynamic configuration to isolate all the tiles in the NoC except for the one(s) involved with the update, thereby minimizing the risk of unauthorized data access or tampering. In this framework, Bastion's security sockets serve as an active distributed extension of the HWRoT, providing functionalities for the secure runtime management of the tiles.

4.4 Secure communications

A secure communication channel between the sSockets and the HWRoT is critical for ensuring the Bastion's overall security. This channel prevents unauthorized access and ensures the confidentiality and integrity of sensitive configuration data exchanged within the system.

NoC architectures often provide multiple data planes, with each plane serving specific purposes. In the open-source ESP platform, six NoC communication data planes are available. Among them, the fifth NoC plane is dedicated to low-bandwidth essential functionalities, such as propagating interrupts and managing configuration registers.

To seamlessly integrate with the existing ESP platform, Bastion leverages the fifth NoC plane to implement secure communication between the HWRoT and the sSockets within the NoC tiles. The security of communications on this plane is validated through a specific security property, as described in Section 5. This ensures that the system maintains robust communication security without introducing overhead or architectural changes.

This approach is generalizable to other NoC backbone architectures. Secure communications can similarly be implemented by leveraging an existing data plane deemed secure,

with its security confirmed through verifiable security properties. If no secure data plane is available, an alternative approach is the introduction of a dedicated data plane for secure communications. For instance, in the ESP platform, a seventh NoC plane could be reserved exclusively for secure interactions between the HWRoT and the sSockets. This would require the addition of corresponding routers and structural modifications to ESP. While this approach enhances isolation and security, it also incurs higher resource costs and complexity in routing.

For completeness, we evaluate the cost and scalability of introducing a dedicated secure data plane in our experimental evaluation, as reported in Section 6.2. This evaluation provides insights into the trade-offs between security and resource utilization in such architectural modifications.

5 Bastion: Security Verification

The hardware security verification process aims to detect and remove any security weaknesses from a given hardware design. This process is complementary to, yet ultimately differs from, functional verification, which ensures a design is functionally correct. Through analyzing source design files, security verification ensures security flaws are not introduced as a result of the way a desired functionality has been implemented.

In the context of Bastion, we individuate a comprehensive security verification framework as fundamental to providing robust guarantees that our design is operating securely, i.e., free from weaknesses that could be exploited by third-party IPs to bypass the access control policy.

We perform a property-driven security verification of the sSocket at three levels of abstraction: security socket level, communications level, and system level. Our security verification approach employs six steps to systematically define a threat model (step 1), identify relevant design assets (step 2), identify potential weaknesses for those assets (step 3), and then verify those weaknesses are not present in the design (steps 4-6).

Identifying all potential weaknesses for a given set of assets is a non-trivial task, even for knowledgeable security verification experts. To ensure coverage of potential weaknesses, we leverage MITRE's hardware Common Weakness Enumeration (CWE) database. Recent work proposed methodologies to automate or semi-automate the selection of relevant CWEs using ML models [ALC+22]. However, these approaches require extensive review and are not exempt from the limitations of ML-based approaches (including hallucinations and potential partial coverage). From this consideration and given the generality of CWEs, in Bastion we opted to manually analyze each CWE in the context of our threat model to individuate the relevant CWEs, using a methodology similar to [AKT+21, Acc]. In particular, we manually review each CWE hardware entry to determine whether the described weakness applies to our design based on relevance to our threat model, set of design assets, and levels of abstraction under analysis. This manual review yields a set of relevant CWEs primarily related to access control and resource sharing. Following, we map each of these CWEs to one or more security properties in our security analysis.

In our security verification framework, we use a combination of hardware information flow tracking (IFT) properties and trace properties. Hardware IFT properties are useful for proving non-interference and other security-relevant design properties [DMR+22]. They are a form of hyperproperty [CS10], which specifies behaviors over sets of execution traces. By contrast, trace properties prove behaviors over a single trace of execution.

```
sig_src when (tagging_condition_expr) =/=> sig_dst /IFT pr.
(sig_src != sig_dst) && tagging_condition_expr     /Trace pr.
```

The IFT property uses the no-flow operator (`=/=>`) to indicate non-interference between

the source signal `sig_src` and the destination signal `sig_dst`. Any time the tagging condition is true, each bit of the source's information will be tagged with a security label. If the tagged information ever propagates to any of the bits of the destination, the IFT property will fail. The verification of an IFT property requires specialized verification tools capable of modeling and tracking information flow in hardware designs. However, trace properties do not require any specialized tools beyond the standard functional verification tools. The example trace property uses standard logical operators to indicate that the source and destination should not be equal when the tagging condition is true. At first glance, it may seem like the two properties are equivalent. But they are not. The trace property cannot discern if the two signals are equal because one was directly assigned to the other or if they happen to have the same value.

5.1 Security socket verification

1.) Define the Threat Model:

We assume that the third-party IP in the tile is untrusted. The NoC backbone is trusted. We evaluate the possibility of any illegal communication occurring between the IP and the NoC backbone. We consider illegal communication to be any transfer of information disallowed by the access control policy configured within the sSocket.

2.) Identify the Assets: An asset is any design resource or component that should be protected under the current threat model. Asset identification facilitates the verification process by constraining the security analysis to a concrete set of critical design elements. At the security-socket level, there are two groups of assets. The first group is the set of signals connecting the sSocket to both the IP and the NoC router. The second group is the set of configuration and control registers containing the sSocket's access control policy.

3.) Identify the Potential Weaknesses: A weakness is any mechanism that could jeopardize the security of the defined assets. We leverage the knowledge contained within Mitre's Common Weakness Enumeration (CWE) database [KRM+22] to guide the analysis towards known access-control-related weaknesses – we identify seventeen CWEs for this level of verification:

Relevant CWEs: 1220, 1221, 1244, 1258, 1259, 1264, 1266, 1267, 1268, 1269, 1270, 1271, 1272, 1274, 1280, 1282, 1326

4.) Define the Security Requirements: The requirements we specify describe failure mechanisms that enable illegal information flow between the IP and the NoC backbone. For assets in group A, we specify that there should only be information flow between the IP and the router if the sSocket deems a request legal. In all other circumstances, the information flowing to the IP and the router should originate from the sSocket and correspond to safe default values.

Requirement 1: The IP cannot receive/send data from/to the router, which originates while the sSocket is in reset mode.

For assets in group B, we ensure that they are cleared/initialized on any state/mode transitions (especially during reset).

Requirement 2: The configuration/control registers are cleared and set to their default values while the sSocket is actively being reset.

5.) Specify the Security Properties: We translate the security requirements into formally specified security properties. Each security property addresses the asset-specific failure mechanism using explicit values, design signals, and operators rather than plain language. For Requirement 1, we formalize it as an Information Flow Property (IFT), which fails if any information between the IP and the NoC backbone flows during reset.

```
‘signal_from_A‘ when (rst == 0) =/=> ‘signal_to_R‘ /Acc->Rout
‘signal_from_R‘ when (rst == 0) =/=> ‘signal_to_A‘ /Rout->Acc
```

We formalize requirement 2 as a trace property which fails if any of the sSocket's configuration and control registers are not configured to safe default values during reset.

```
‘reg‘ == ‘dflt_val‘ unless (rst != 0 && ‘socket_state‘ != 0)
```

6.) Verify the Security Properties: We use commercial hardware verification tools to determine whether each security property holds. We utilize Siemens QuestaSim to simulate a complete ESP SoC both with and without Bastion. In both simulations, we bind and co-simulate a security model based on the RTL of the SoC and constructed by Cycuity Radix-S [Rad] with a set of Cycuity security rules assertions. Each security rule assertion correlates with one of the expanded security properties. The Radix-S security model reports how many times each individual property assertion fails along with the time at which each failure occurs.

5.2 Communications verification

At the firmware level, the security verification process focuses on securing the communication between the sSocket and the trusted entity (TE) which configures the access control policy in the sSocket's registers.

1.) Define the Threat Model: Section 5.1 is still applicable at this level, with the additional assumption that the configuration coming from the TE over the security NoC plane is trusted.

2.) Identify the Assets: We focus on the set of configuration registers containing the sSocket access control policy.

3.) Identify the Potential Weaknesses: We identify seven CWEs relevant to the task of configuring, managing, and preserving the confidentiality and integrity of critical registers.

Relevant CWEs: 276, 1191, 1193, 1262, 1283, 1290, 1292

4.) Define the Security Requirements: We specify that the sSocket's configuration registers must only be set by the TE via the designated security configuration plane.

Requirement 3: The configuration registers contain the default values until they are modified by the TE (config.).

5.) Specify the Security Properties: We formalize Requirement 3 as an IFT property that fails if any information other than the TE-provided configuration flows into the registers.

```
‘unauthorized_signal‘ when (‘reg‘ == ‘dflt_val‘)
 =/=>
‘reg‘ unless (‘reg‘ == ‘dflt_val‘)
```

6.) Verify the Security Properties: We utilize the same verification setup and procedure described in Section 5.1.

5.3 System-level verification: avoid proxy scenarios

At the system level, the security verification process focuses on identifying any unintended proxy scenarios that would enable an IP to send/receive data to/from an unauthorized memory region. For a given IP, an unauthorized memory region is any region whose address does not reside within the range of legal addresses configured in the local sSocket.

1.) Define the Threat Model: Same as Section 5.1.

2.) Identify the Assets: We focus on the set of signals connecting the sSocket to both the IP and the NoC router.

3.) Identify the Potential Weaknesses: We identify three CWEs relevant to system-level properties involving unintended proxy scenarios, resource sharing between trusted and untrusted entities, and overlapping access control policies.

Relevant CWEs: 441, 1189, 1260

4.) Define the Security Requirements: We specify that each IP in the system must only be able to access those addresses contained within its sSocket's configured access control policy.

Requirement 4: Any IP cannot receive/send data from/to any address not contained within its sSocket's access control policy.

5.) Specify the Security Properties: We formalize Requirement 4 as an IFT property that fails if any information flows between an unauthorized IP and a protected address.

```
‘sig_from_A‘ =/=> ‘unauthorized_resource‘
‘sig_from_resource‘ when (address == ‘unauthorized_address‘)
=/=>
‘sig_to_A‘
```

6.) Verify the Security Properties: We follow the same verification setup and procedure described in Section 5.1.

6 Experimental validation

This section aims at experimentally evaluating Bastion, comparing performance and resource consumption of ESP Bastion-enhanced SoCs with the corresponding standard ESP-based SoC. In Section 6.1 we evaluate the resource consumption, timings, and power performance of the sSocket introduced in Section 4.2. Section 6.2 showcases the overall system-level performance on realistic use-case scenarios.

6.1 sSocket: resources, timings, and power estimation

6.1.1 Experimental setup

This first set of experiments evaluates the resource consumption, power impact, and timing closure for the sSocket proposed in Bastion. The results are obtained using AMD Vivado 2022.2 implementing the designs targeting a VCU118 development board equipped with a Virtex Ultrascale+ FPGA.

Table 2: Resource, power, and worst negative slacks of the ESP standard socket compared with the ESP standard Socket + BASTION. Both ESP Std Socket and ESP + sSocket designs meet timings in all of the tested scenarios - a higher WNS does not necessarily indicate faster performance; it only means more margin to the target timing constraints.

		LUTs	**FFs**	**Power**	**WNS (ns)**
100 MHz	ESP Std Socket	3910	6695	0.612	7.464
	ESP + sSocket	4214	7045	0.612	5.687
187 MHz	ESP Std Socket	3910	6695	0.629	3.187
	ESP + sSocket	4217	7045	0.628	1.792
250 MHz	ESP Std Socket	3911	6695	0.640	2.046
	ESP + sSocket	4233	7063	0.639	0.700
300 MHz	ESP Std Socket	3910	6695	0.650	1.552
	ESP + sSocket	4285	7063	0.652	0.357

6.1.2 Results

Table 2 compares the results of the ESP standard socket (ESP Std Socket) and the ESP socket enhanced with the sSocket functionalities (ESP + sSocket) across various target frequencies. The experiments highlight three key outcomes:

- *Resource Consumption:* the integration of the sSocket incurs a limited increase in resource utilization. Specifically, the sSocket results in at most a 9.5% increase in LUTs and a 5.5% increase in FFs compared to the standard ESP socket. This underscores the efficiency of the sSocket design and its suitability for integration in resource-constrained systems, as also demonstrated in the system-level resource utilization evaluation discussed in Section 6.2.

- *Power Estimation:* the power consumption of the ESP socket remains nearly unaffected by the inclusion of the sSocket. Across all tested configurations, the power differences are minimal, demonstrating the low power overhead of the proposed design. This result confirms that the sSocket can be integrated into existing systems without adversely affecting their power consumption.

- *Timing Closure:* The ESP standard socket enhanced with BASTION's sSocket successfully meets timing constraints across all tested frequencies. AMD Vivado was able to correctly close the timing in all of the tested scenarios, demonstrating that the sSocket does not considerably impact the critical path in the standard ESP socket.

The experiments underscore the lightweight nature of the sSocket in terms of resource usage, power efficiency, and performance impact. The results demonstrate that the sSocket can provide robust security without compromising the operational efficiency of the ESP standard socket, further validating its practicality and scalability for deployment in real SoC applications.

6.2 Case-study: system-level performance and area impact

This set of experiments demonstrates the performance impact, resource consumption, scalability, and effectiveness of BASTION on realistic use-case scenarios. We leverage the ESP framework to deploy realistic sample SoC architectures targeting edge applications that require efficient hardware acceleration of matrix multiplication operations, such as image processing and AI workloads. We aim with this experiment to test BASTION on realistic benchmarks and evaluate integration and performance.

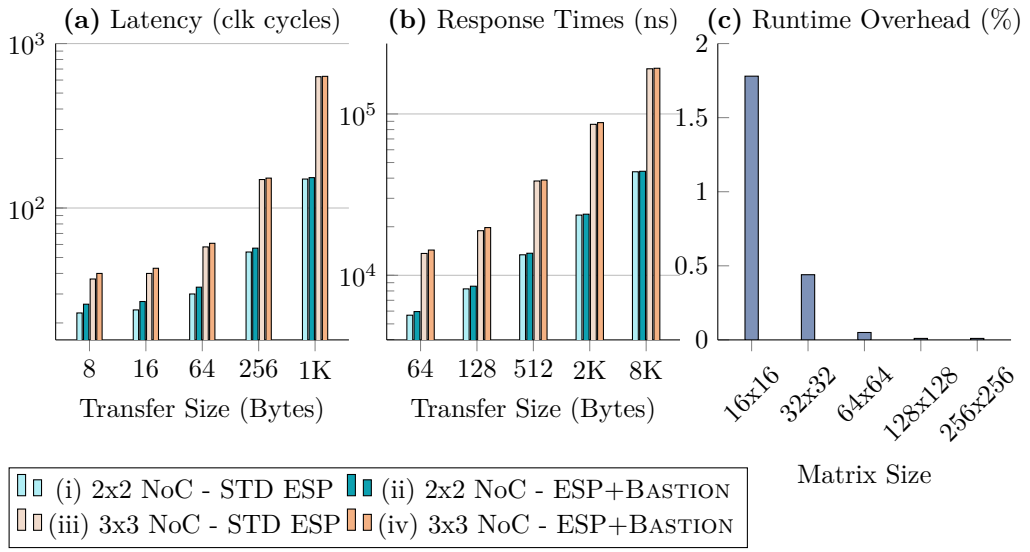

Figure 3: Performance of the standard ESP platform (STD ESP) compared with the ESP platform enhanced with Bastion (ESP+Bastion). (a) 3PIP (ESP third-party workflow); (b) GEMM accelerator (ESP accelerator workflow). In (c) the maximum measured overhead of Bastion is reported as a function of different GEMM accelerator structures (in percentage).

6.2.1 Experimental setup

We deployed four baseline SoCs ((i), (ii), (iii), and (iv)) each featuring three fixed tiles: a DRAM memory tile, a RISC-V CVA6 processor core tile, and an I/O and HWRoT tile (security tile). To evaluate Bastion's performance and scalability, we tested two NoC mesh integrating two additional accelerator IPs considered to be untrusted:

- a generic third-party traffic generator IP (3PIP), integrated using the ESP third-party IP flow.
- a General Matrix Multiplication (GEMM) accelerator, developed using Cadence Stratus HLS and integrated with the standard ESP flow.

The baseline SoCs are configured as follows:

- SoC (i): a standard ESP NoC structure with an additional tile integrating the 3PIP accelerator.
- SoC (ii): the same structure as SoC (i), enhanced with Bastion.
- SoC (iii): a standard ESP NoC structure with an additional tile integrating the GEMM accelerator.
- SoC (iv): the same structure as SoC (iii), enhanced with Bastion.

All configurations are synthesized and implemented using AMD Vivado 2022.2 and deployed on an AMD VCU118 development board equipped with a Virtex Ultrascale+ FPGA.

By considering different accelerator IPs, this setup allowed us to analyze Bastion's performance across multiple configurations and data patterns. The experiments emphasized Bastion 's adaptability to different architectures, highlighting its ability to seamlessly integrate with various IPs and maintain robust security guarantees.

Table 3: Overall area impact of Bastion on an ESP system deployed for the AMD VCU118 FPGA development board considering the single accelerator tile, a full 2x2 SoC, and the cost of the optional additional NoC security data plane.

2x2 ESP NoC	**LUTs**	**FFs**	**BRAM**	**DSP**
Accelerator Tile (Baseline)	24,755	20,501	19	45
Accelerator Tile + sSocket	25,130	20,868	19	45
Percent Increase	**1.51%**	**1.79%**	**-**	**-**
Full SoC (Baseline)	114,946	106,470	201	72
Full SoC + Bastion	116,446	107,161	201	72
Percent Increase	**1.31%**	**0.65%**	**-**	**-**
Optional NoC security data plane	2179	1968	-	-
Percent Increase (Full SoC)	**8.8%**	**9.6%**	**-**	**-**

6.2.2 Performance

We leverage the RISC-V processor to set up different data movements for the GEMM accelerator and the 3PIP. The GEMM accelerator and the 3PIP are invoked by bare-metal software running on the CVA6 core integrated into the processor tile of the SoC. Once configured and started, both the GEMM accelerator and the 3PIP are capable of independently fetching data from the DRAM memory integrated into the memory tile of the SoC by issuing DMA requests.

Figure 3(a) reports the latency (in clock cycles) measured for the 3PIP as a function of the transfer size initiated by 3PIP and directed to the DRAM tile. Considering all configurations, the highest measured impact of Bastion is 4.9% (transfer size 8 bytes). This impact decreases with longer transfers – the measured impact on the 1KB transfer is 0.7%.

This trend is also confirmed in Figure 3(b). In this second scenario, the performance overhead of Bastion is evaluated by running matrix multiplication workloads of various sizes on the GEMM accelerator, generating data movements ranging from 64 bytes to 8 KB still directed to the DRAM memory tile through the NoC.

Figure 3(c) shows the latency impact (in percentage) of adding Bastion to the GEMM tile and executing matrix multiplications ranging from 16x16 (3KB workload) to 256x256 (768 KB workload). As observed in the data, the performance overhead is limited across all tested configurations. The highest impact is measured for the 16x16 workload configuration (1.78% overall latency impact). The impact decreases as the workload size increases, dropping below 0.01% for the 128x128 workload configuration and larger.

This trend arises because issuing long burst transactions allows to reduce the signaling overhead – the cost of analyzing the address is amortized over a long burst of data, resulting in a reduced performance impact concerning shorter bus transactions. Leveraging long burst transactions to sequential blocks of data is a typical methodology used to enhance communication performance in high-performance streaming accelerators and minimize the impact of Bastion on the system's performance.

6.2.3 Resource consumption

Table 3 compares the FPGA resource utilization in terms of Look-Up Tables (LUTs), Flip-Flops (FFs), Block RAM (BRAM), and Digital Signal Processing (DSP) slices between the baseline 2x2 NoC SoC (SoC (i) in the previous experiment) and the 2x2 NoC SoC equipped with Bastion (SoC (ii) in the previous experiment) and reports the cost of the optional additional NoC security data plane. The table reports, first, the resource overhead introduced by a single security socket (sSocket) on a single NoC tile. Following this, the total resource cost of deploying the overall Bastion-based access control system across

the full SoC. Finally, the overall cost and impact of adding an additional data plane for secure communications. To better showcase the impact of Bastion on the NoC backbone, we do not consider the HWRoT in these measurements.

The integration of a sSocket in a single tile results in a modest resource impact, increasing LUT usage by 1.51% and FF usage by 1.79%. Bastion does not require any additional BRAM or DSP resources, leaving the usage of these resources unchanged. When considering the overall impact of Bastion across the entire 2x2 NoC-based SoC, the resource impact is still limited: 1.31% for LUTs and less than 1% for FFs. Again, there is no increase in BRAM or DSP utilization.

These results highlight Bastion's scalability and minimal resource overhead, ensuring that it can be seamlessly integrated into existing NoC-based architectures without imposing significant extra area costs. The lightweight nature of Bastion makes it particularly suitable for resource-constrained platforms.

Additionally, as introduced in Section 4.4, when required, Bastion could be supported by introducing a security NoC data plane dedicated to secure communication. In the ESP framework, this would be the 7th data plane in the system. The addition of this plane would require a corresponding router in each tile. According to our results, each NoC router for the data plane utilizes 2.2% of LUTs and 2.4% of FFs per tile. Considering the whole SoC, the cost of the seventh NoC data plane is 8.8% in LUT and 9.6% in FF.

Considering the ESP framework, the NoC backbone is expected to require minimal architectural modifications to accommodate the extra data plane. However, this change might require more complex architectural modifications in other NoC-based platforms. From the previous considerations, the impact of adding an additional data plane is not negligible and should be carefully evaluated when security requirements do not allow the use of a plane natively available in the NoC backbone under analysis.

6.2.4 Effectiveness of Bastion

In this section, we aim to demonstrate the effectiveness of Bastion by considering realistic workloads executed on the SoC (i) and (ii) previously introduced, integrating the 3PIP. In this scenario, the 3PIP is configured to attempt illegal transactions to private data structures, We aim to measure Bastion's ability to proactively stop and report these attempts to the HWRoT.

We assume the CVA6 processor to have one or multiple private data structures stored in the shared DRAM memory tile. We simulate a DMA attack by configuring the 3PIP to issue illegal transactions aimed at fetching data (reading) and compromising (writing) one of the processor's private structures in DRAM. In a deployed platform, these illegal transactions could originate from a malicious attack, local misbehavior, or faults of the 3PIP.

After configuration and execution start, in SoC (i), as expected, we observe that both read and write illegal transactions are propagated to the NoC backbone until reaching and being served by the DRAM memory tile. This occurs because the baseline ESP framework does not deploy access control systems. Consequently, the 3PIP can freely access and modify the processor's private data, thereby compromising the confidentiality and integrity of the processor's private data structures.

We conduct the same experiment using SoC (ii) enhanced with Bastion. We configure the local sSocket to supervise the 3PIP enforcing least-privilege access, defining a local access control list that allows access rights only to the memory area assigned to the 3PIP tile and legitimately shared with the processor.

As expected and confirmed by our security verification, we observe that the sSocket detects the attempts of the 3PIP to access the processor's private area and proactively stops any illegal transaction from entering the NoC backbone, thereby protecting the system's integrity and confidentiality from the compromised IP. After detection, an interrupt is

generated to the HWRoT, allowing the HWRoT to take further actions against the 3PIP. This experimentally confirms the effectiveness of Bastion and its capability to proactively isolate malicious third-party IPs.

7 Discussion

The primary goal of Bastion is to provide a comprehensive, verifiable, and high-performance access control security framework for NoC-based SoC architectures, particularly tailored for open-source platforms. Although our solution builds on some prior works, to our knowledge, no previous work has offered an open-source framework that combines design, verification, and secure communication with HWRoT for NoCs – with this work, we focus on enhancing the security of open-source NoC platforms, an area with only limited prior contributions in the hardware-security community. Our aim is to raise the security bar in open-source platforms and democratize access to more secure SoCs for the research community and industry. While access control mechanisms are well-studied, Bastion introduces several unique aspects that distinguish it from existing solutions. Following, we summarize the main features of Bastion:

- *Rigorous Security Verification:* Bastion combines a design process with a rigorous property-based security verification process tailored to address the hardware MITRE CWEs, providing a systematic and formal approach to mitigate security weaknesses. This methodology ensures robust security guarantees by addressing weaknesses that can be overlooked in traditional verification processes.
- *Integration with Open-Source Platforms:* unlike proprietary or closed frameworks, Bastion is specifically designed to integrate seamlessly with open-source platforms like ESP. With this integration, we aim to enhance the security of these platforms, support the open-hardware community's broader goals of transparency and trustworthiness, and foster collaboration and innovation in the open-source hardware security domain.
- *Distributed Decision-Making:* Bastion supports the development of distributed access control systems. In our vision, Bastion access control systems act as an extension of the HWRoT. The HWRoT is the centralized trusted source for the configuration and management of the access control lists in the sSockets. Bastion 's sSockets enforce access control decisions locally after configuration, enabling distributed decision-making to reduce the reliance on a single point of failure, enhancing scalability, and minimizing the latency associated with the security checks.
- *Dynamic Policy Management:* Bastion supports runtime reconfiguration of access control lists. This capability provides flexibility in dynamic systems that must adapt to changing operational requirements.
- *Extendibility:* Bastion is conceived to be easily extensible. The sSocket design can be extended with additional functionalities related, for instance, to capability-like access control, bandwidth management, prevention of Denial-of-Service attacks, and other advanced mechanisms. Similarly, Bastion's property-based security verification can be extended by adding security features targeting additional security services or specific target applications.
- *Portability:* we designed Bastion to be portable across a wide range of NoC-based platforms, including both open-source and commercial SoC designs. Its modular architecture, coupled with its operation directly at the NoC flit level, ensures transparent operation and compatibility with all of the communication protocols supported by the NoC backbone.

7.1 Current limitations

Bastion aims to establish a step forward in the development of secure and verifiable open-source security solutions for NoC-based platforms by addressing gaps in existing platforms and providing robust security guarantees. This approach extends the applicability of open-source platforms to critical and high-stakes domains.

However, at its current stage, Bastion does not include functionalities like capability-based access control features [WWN+15]. These features would enable access control at the even finer granularity of the individual NoC transaction, offering greater flexibility and adaptability in defining access control lists. Capabilities-based control systems introduce non-trivial challenges in NoC-based platforms, such as managing the overhead and ensuring efficient validation, which can significantly impact system performance and scalability. Considering these trade-offs, Bastion is designed as a general framework with minimal performance impact, while remaining adaptable for extensions like capability-based access control in applications where the added granularity and flexibility justify the associated performance and scalability costs.

In future work, we plan to explore these tradeoffs by integrating capability-based access control in Bastion. This extension will involve implementing token-based permissions, where capability tokens define precise access rights to specific resources. The existing infrastructure of Bastion is well-suited for this integration: tokens can be securely generated, distributed, and managed by the hardware root of trust via the secure communication channel, and validated by an enhanced sSocket equipped with a dedicated Capabilities Table. Bastion's security verification framework can be seamlessly expanded to validate the functionality and security of the capabilities mechanism.

Bastion does not prevent Denial-of-Service [RK22] or memory flood attacks of legal transactions [RPB+19]. As mentioned in Section 2.3, these attacks are currently outside of our threat model. We intentionally focus our work on access control rather than covering multiple attacks/countermeasures to keep the focus on providing a complete baseline design and verification methodology extensible for the development of further security countermeasures. In our future works, we plan to focus on the integration and security validation of further defensive methodologies aiming at covering other security challenges including: Denial-of-Service attacks [RK22], secure bandwidth management [BOB+24, RBM+20], and side-channel attacks and fault injection [WGO+13, WS12, SPH+21].

We note that, for implementing our experimental campaign we employed some proprietary tools like AMD's Vivado suite for FPGA synthesis and implementation and Siemens QuestaSIM for simulations. However, our framework itself does not rely on any vendor-specific features or closed-source infrastructure. Bastion is fully described in standard hardware description languages, making it equally compatible with popular open-source tools, including Verilator [Verb, Vera] (simulations), the Yosys toolchain [SHW+19, Yos] (synthesis and Verilog to bitstream flow for FPGAs), and the OpenROAD/OpenLane toolchain [KS21, Opea] (RTL to GDS flow).

8 Conclusion

We presented Bastion, an open-source framework for the secure integration of third-party IPs on NoC-based platforms. Bastion aims to bridge the gap between open-source accessibility, trustworthiness, and robust security guarantees. The integration of our framework with a mature open-source hardware platform, such as ESP, introduces provably verified features aiming at enforcing the system security of critical applications. By combining a robust design, comprehensive security verification, and a methodology for the secure configuration and management of the access control system, Bastion aims to support the application of open-source platforms in critical domains requiring strong

security guarantees.

Our experimental evaluation demonstrated the practical effectiveness of BASTION through simulated experiments and FPGA implementations of realistic SoCs. These evaluations confirmed that the framework has a minimal impact on performance and resource consumption. This highlights the feasibility of integrating advanced security features without significantly affecting the overall efficiency and resource utilization of the platform.

We designed BASTION with modularity in mind, ensuring that our design and verification methodology can be easily extended to target further security challenges. For instance, future extensions will focus on incorporating methodologies for capability-based access control, bandwidth management, resource isolation, or other critical services, further enhancing the framework's functionalities.

Looking ahead, we plan to release BASTION as an open-source project. This goes in the direction of promoting transparency and accessibility and stimulating further research efforts in security for open-source hardware. By encouraging the adoption of rigorous security verification techniques, we aim to support the deployment of open-source platforms in critical applications including automotive, medical devices, and space, among others.

References

[ABG+20] Alon Amid, David Biancolin, Abraham Gonzalez, Daniel Grubb, Sagar Karandikar, Harrison Liew, Albert Magyar, Howard Mao, Albert Ou, Nathan Pemberton, Paul Rigge, Colin Schmidt, John Wright, Jerry Zhao, Yakun Sophia Shao, Krste Asanović, and Borivoje Nikolić. Chipyard: Integrated design, simulation, and implementation framework for custom socs. *IEEE Micro*, 40(4):10–21, 2020.

[Acc] Accelera. *Security Annotation for Electronic Design Integration (SA-EDI) 1.0, 2021-07-13, www.accellera.org/downloads/standards/ip-security-assurance.*

[AKT+21] Sohrab Aftabjahani, Ryan Kastner, Mark Tehranipoor, Farimah Farahmandi, Jason Oberg, Anders Nordstrom, Nicole Fern, and Alric Althoff. Special session: Cad for hardware security - automation is key to adoption of solutions. In *2021 IEEE 39th VLSI Test Symposium (VTS)*, pages 1–10, 2021.

[ALC+22] Baleegh Ahmad, Wei-Kai Liu, Luca Collini, Hammond Pearce, Jason M Fung, Jonathan Valamehr, Mohammad Bidmeshki, Piotr Sapiecha, Steve Brown, Krishnendu Chakrabarty, et al. Don't cweat it: Toward cwe analysis techniques in early stages of hardware design. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, pages 1–9, 2022.

[ARM] ARM. *ARM CoreLink TZC-400 TrustZone Address Space Controller Technical Reference Manual.*

[BOB+24] Thomas Benz, Alessandro Ottaviano, Robert Balas, Angelo Garofalo, Francesco Restuccia, Alessandro Biondi, and Luca Benini. Axi-realm: A lightweight and modular interconnect extension for traffic regulation and monitoring of heterogeneous real-time socs. In *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6, 2024.

[BPOD14] J. Brunel, R. Pacalet, S. Ouaarab, and G. Duc. Secbus, a software/hardware architecture for securing external memories. In *2014 2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, 2014.

[CCGD12] P. Cotret, J. Crenne, G. Gogniat, and J. Diguet. Bus-based mpsoc security through communication protection: A latency-efficient alternative. In *2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*, pages 200–207, 2012.

[CGR+24] Francesco Conti, Angelo Garofalo, Davide Rossi, Giuseppe Tagliavini, and Luca Benini. Open-source heterogeneous socs for ai: The pulp platform experience. *arXiv preprint arXiv:2412.20391*, 2024.

[CS10] Michael R Clarkson and Fred B Schneider. Hyperproperties. *Journal of Computer Security*, 18(6):1157–1210, 2010.

[DMR+22] Calvin Deutschbein, Andres Meza, Francesco Restuccia, Mattew Gregoire, Ryan Kastner, and Cynthia Sturton. Toward hardware security property generation at scale. *IEEE Security and Privacy*, 2022.

[FPL+08] Leandro Fiorin, Gianluca Palermo, Slobodan Lukovic, Valerio Catalano, and Cristina Silvano. Secure memory accesses on networks-on-chip. *IEEE Transactions on Computers*, 57(9):1216–1229, 2008.

[GCE+20] Davide Giri, Kuan-Lin Chiu, Guy Eichler, Paolo Mantovani, N Chandramoorth, and Luca P Carloni. Ariane+ nvdla: Seamless third-party ip integration with esp. In *Workshop on Computer Architecture Research with RISC-V (CARRV)*, 2020.

[GPP+15] Miltos D Grammatikakis, Kyprianos Papadimitriou, Polydoros Petrakis, Antonis Papagrigoriou, George Kornaros, Ioannis Christoforakis, Othon Tomoutzoglou, George Tsamis, and Marcello Coppola. Security in mpsocs: a noc firewall and an evaluation framework. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(8):1344–1357, 2015.

[HPSK06] Ted Huffmire, Shreyas Prasad, Tim Sherwood, and Ryan Kastner. Policy-driven memory protection for reconfigurable hardware. In *European Symposium on Research in Computer Security*. Springer, 2006.

[KAA23] Emre Karabulut, Amro Awad, and Aydin Aysu. Ss-axi: Secure and safe access control mechanism for multi-tenant cloud fpgas. In *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2023.

[KNRSV00] Kurt Keutzer, A Richard Newton, Jan M Rabaey, and Alberto Sangiovanni-Vincentelli. System-level design: Orthogonalization of concerns and platform-based design. *IEEE transactions on computer-aided design of integrated circuits and systems*, 19(12):1523–1543, 2000.

[KRM+22] Ryan Kastner, Francesco Restuccia, Andres Meza, Sayak Ray, Jason Fung, and Cynthia Sturton. Automating hardware security property generation. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pages 1384–1387, 2022.

[KS21] Andrew B Kahng and Tom Spyrou. The openroad project: Unleashing hardware innovation. 2021.

[MGDG+20] Paolo Mantovani, Davide Giri, Giuseppe Di Guglielmo, Luca Piccolboni, Joseph Zuckerman, Emilio G Cota, Michele Petracca, Christian Pilato, and Luca P Carloni. Agile soc development with open esp. In *Proceedings of the 39th International Conference on Computer-Aided Design*, pages 1–9, 2020.

[MIT] MITRE. *The CWE Official Webpage*. https://cwe.mitre.org/.

[MKAA24] Arsalan Ali Malik, Emre Karabulut, Amro Awad, and Aydin Aysu. Enabling secure and efficient sharing of accelerators in expeditionary systems. *Journal of Hardware and Systems Security*, pages 1–19, 2024.

[MRO+23] Andres Meza, Francesco Restuccia, Jason Oberg, Dominic Rizzo, and Ryan Kastner. Security verification of the opentitan hardware root of trust. *IEEE Security & Privacy*, 21(3):27–36, 2023.

[Opea] *The OpenROAD project offical Github*. https://github.com/the-openroad-project.

[Opeb] *The OpenTitan official website*. https://opentitan.org/.

[PGC22] Luca Piccolboni, Davide Giri, and Luca P Carloni. Accelerators & security: The socket approach. *IEEE Computer Architecture Letters*, 21(2):65–68, 2022.

[PSC+24] Arpan Suravi Prasad, Moritz Scherer, Francesco Conti, Davide Rossi, Alfio Di Mauro, Manuel Eggimann, Jorge Tomás Gómez, Ziyun Li, Syed Shakib Sarwar, Zhao Wang, Barbara De Salvo, and Luca Benini. Siracusa: A 16 nm heterogenous risc-v soc for extended reality with at-mram neural engine. *IEEE Journal of Solid-State Circuits*, 59(7):2055–2069, 2024.

[Rad] *The Cycuity Radix-S offical website*. https://cycuity.com/solutions/.

[RBFSG22] Jan Richter-Brockmann, Jakob Feldtkeller, Pascal Sasdrich, and Tim Güneysu. VERICA - verification of combined attacks: Automated formal verification of security against simultaneous information leakage and tampering. Cryptology ePrint Archive, Paper 2022/484, 2022.

[RBM+20] Francesco Restuccia, Alessandro Biondi, Mauro Marinoni, Giorgiomaria Cicero, and Giorgio Buttazzo. Axi hyperconnect: A predictable, hypervisor-level interconnect for hardware accelerators in fpga soc. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2020.

[RCSP23] Manuel Rodríguez, Francisco Costa, Bruno Vilaça Sá, and Sandro Pinto. Open-source risc-v input/output memory management unit (iommu) ip. 2023.

[RK22] Francesco Restuccia and Ryan Kastner. Cut and forward: Safe and secure communication for fpga system on chips. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(11):4052–4063, 2022.

[RMK21] Francesco Restuccia, Andres Meza, and Ryan Kastner. Aker: A design and verification framework for safe and secure soc access control. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9. IEEE, 2021.

[RMKO22] Francesco Restuccia, Andres Meza, Ryan Kastner, and Jason Oberg. A framework for design, verification, and management of soc access control systems. *IEEE Transactions on Computers*, 72(2):386–400, 2022.

[RPB+19] Francesco Restuccia, Marco Pagani, Alessandro Biondi, Mauro Marinoni, and Giorgio Buttazzo. Is your bus arbiter really fair? restoring fairness in axi interconnects for fpga socs. *ACM Transactions on Embedded Computing Systems (TECS)*, 18(5s):1–22, 2019.

[RSM+24] Jayden Rogers, Niyaz Shakeel, Divya Mankani, Samantha Espinosa, Cade Chabra, Kaki Ryan, and Cynthia Sturton. Security properties for open-source hardware designs. *arXiv preprint arXiv:2412.08769*, 2024.

[SAHS+18] Johanna Sepulveda, Damian Aboul-Hassan, Georg Sigl, Bernd Becker, and Matthias Sauer. Towards the formal verification of security properties of a network-on-chip router. In *2018 IEEE 23rd European Test Symposium (ETS)*, 2018.

[SBP+25] Paul Scheffler, Thomas Benz, Viviane Potocnik, Tim Fischer, Luca Colagrande, Nils Wistoff, Yichao Zhang, Luca Bertaccini, Gianmarco Ottavi, Manuel Eggimann, Matheus Cavalcante, Gianna Paulin, Frank K. Gürkaynak, Davide Rossi, and Luca Benini. Occamy: A 432-core dual-chiplet dual-hbm2e 768-dp-gflop/s risc-v system for 8-to-64-bit dense and sparse computing in 12-nm finfet. *IEEE Journal of Solid-State Circuits*, pages 1–15, 2025.

[SHS18] Fahad Siddiqui, Matthew Hagan, and Sakir Sezer. Pro-active policing and policy enforcement architecture for securing mpsocs. In *2018 31st IEEE International System-on-Chip Conference (SOCC)*. IEEE, 2018.

[SHW+19] David Shah, Eddie Hung, Clifford Wolf, Serge Bazanski, Dan Gisselquist, and Miodrag Milanovic. Yosys+nextpnr: An open source framework from verilog to bitstream for commercial fpgas. In *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 1–4, 2019.

[SPH+21] Amin Sarihi, Ahmad Patooghy, Mahdi Hasanzadeh, Mostafa Abdelrehim, and Abdel-Hameed A Badawy. Securing network-on-chips via novel anonymous routing. In *Proceedings of the 15th IEEE/ACM International Symposium on Networks-on-Chip*, pages 29–34, 2021.

[TEF+20] Benjamin Tan, Rana Elnaggar, Jason M Fung, Ramesh Karri, and Krishnendu Chakrabarty. Towards hardware-based ip vulnerability detection and post-deployment patching in systems-on-chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020.

[TL17] Anna Trikalinou and Dan Lake. Taking DMA attacks to the next level. *BlackHat USA*, pages 22–27, 2017.

[Vera] *The Verilator offical Github*. https://github.com/verilator/verilator.

[Verb] *The Verilator offical website*. https://www.veripool.org/verilator/.

[WGO+13] Hassan MG Wassel, Ying Gao, Jason K Oberg, Ted Huffmire, Ryan Kastner, Frederic T Chong, and Timothy Sherwood. Surfnoc: a low latency and provably non-interfering approach to secure networks-on-chip. *ACM SIGARCH Computer Architecture News*, 2013.

[WS12] Yao Wang and G Edward Suh. Efficient timing channel protection for on-chip networks. In *2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip*, pages 142–151. IEEE, 2012.

[WWN+15] Robert N.M. Watson, Jonathan Woodruff, Peter G. Neumann, Simon W. Moore, Jonathan Anderson, David Chisnall, Nirav Dave, Brooks Davis, Khilan Gudka, Ben Laurie, Steven J. Murdoch, Robert Norton, Michael Roe, Stacey Son, and Munraj Vadera. Cheri: A hybrid capability-system

architecture for scalable software compartmentalization. In *2015 IEEE Symposium on Security and Privacy*, pages 20–37, 2015.

[Yos] *The Yosys project offical Github.* https://github.com/YosysHQ/yosys.

[ZCF24] Feng Zhou, Hua Chen, and Limin Fan. Prover - toward more efficient formal verification of masking in probing model. Cryptology ePrint Archive, Paper 2024/1202, 2024.

[ZRMH+24] Melisande Zonta-Roudes, Andres Meza, Nora Hinderling, Lucas Deutschmann, Francesco Restuccia, Ryan Kastner, and Shweta Shinde. expect: On the security implications of violations in axi implementations. 2024.